Labelled Graph Strategic Rewriting for Social Networks

Maribel Fernández, Hélène Kirchner, Bruno Pinaud, Jason Vallet

Labelled Graph Strategic Rewriting for Social Networks

Maribel Fernández[a], Hélène Kirchner[b,*], Bruno Pinaud[c], Jason Vallet[c]

[a] *King's College London, UK*
[b] *Inria, France*
[c] *University of Bordeaux, CNRS UMR5800 LaBRI, France*

Abstract

We develop an algebraic approach, based on labelled-graph strategic rewriting, for the study of social networks, specifically network generation and propagation mechanisms. This approach sheds a new light on these problems, and leads to new or improved generation and propagation algorithms. We argue that relevant concepts are provided by three ingredients: labelled graphs to represent networks of data or users, rewrite rules to describe concurrent local transformations, and strategies to express control. We show how these techniques can be used to generate random networks that are suitable for social network analysis, simulate different propagation mechanisms, and analyse and compare propagation models by extracting common rules and differences, thus leading to improved algorithms. We illustrate with examples the flexibility of the approach.

Keywords: Labelled port graph, graph rewriting, strategies, strategic rewrite programs, social networks, generation, propagation

1. Introduction

Social networks, representing connected users and their relationships, have been intensively studied in the last years [1, 2, 3]. Their analysis raises several

*Corresponding author
Email addresses: maribel.fernandez@kcl.ac.uk (Maribel Fernández), helene.kirchner@inria.fr (Hélène Kirchner), bruno.pinaud@u-bordeaux.fr (Bruno Pinaud), jason.vallet@u-bordeaux.fr (Jason Vallet)

questions, in particular regarding their construction and evolution. Network propagation mechanisms, replicating real-world social network phenomena, such as the relaying of information announcing events, have applications in various domains, ranging from sociology [4] to epidemiology [5, 6] or even viral marketing and product placement [7].

To gain a better understanding of these phenomena, we need to model and analyse such systems and deal with features that are complex (since they involve data that are massive and highly heterogeneous), dynamic (due to interactions, time, external or internal evolutions), and distributed in networks. We argue in this paper that relevant concepts to address these challenges are: **Labelled Graphs** to represent networks of data or objects, **Rules** to deal with concurrent local transformations, and **Strategies** to control the application of rules (including probabilistic application) and to focus on points of interest. Indeed, since we have to deal with large networks, the ability to perform pattern-matching and to define appropriate views (to focus on areas of interest) is essential. Since data are often corrupted or imprecise, we also need to deal with uncertainty, which implies that we need to be able to address probabilistic or stochastic issues in the models. The dynamic evolution of data is generally modelled by simple transformations, applied in parallel and triggered by events or time. However, such transformations may be controlled by some laws that induce a global behaviour. Modelling may reveal conflicts, which can be detected by computing overlaps of rules and solved, for instance, by using precedence (or other choices). Thus, the ability to define strategies is also essential, including mechanisms to deal with backtracking and history through notions of derivations or traces. Last but not least, visualisation is important at all levels: for data analysis, program engineering, program debugging, tests and verification (for instance to provide proof intuition).

Preliminary results obtained by applying this approach to the study of network propagation phenomena and network generation are described in [8, 9], respectively. In this paper, we recall the main results and build on them to develop more detailed analyses of network generation algorithms, and new and

improved propagation algorithms.

We start by presenting a framework, based on directed port graph rewriting, to define social network models and specify their dynamic behaviour. Port graph rewriting systems have been used to model systems in other domains (e.g., biochemistry, interaction nets, games; see [8, 10, 11, 12]). Here, we adapt to the specific domain of social network modelling the general port graph rewriting notions given in [13], and enrich them to facilitate the implementation of algorithms in this area. Most notably, here we use oriented edges and add a primitive notion of conditional matching.

We use this framework to specify an algorithm to generate networks that have the characteristics of real-world social networks, and to study the process of propagation (of actions such as relaying information, spreading gossip, etc.) within a social network. For the latter, we consider three different propagation algorithms: the *independent cascade* model **IC** [14], the *linear threshold* model **LT** [15], and the *Riposte* model **RP** [16]. The first two propagation models were specified using strategic rewriting in [8]. In this paper, we show that these two models are instances of the same strategic graph program, and we also develop a strategic rewriting definition of **RP**. This latter model can be seen as an improved version of **IC**, although with slight differences, where the propagation process ensures that users' opinions will not be disclosed to passive observers. Based on this idea, we also develop a new propagation model: a privacy-aware version of **LT**, which we call **RP-LT**.

All algorithms have been implemented in PORGY, an interactive modelling tool based on port graph rewriting; more details concerning the rewriting platform can be found in [17]. Furthermore, since the instructions used in PORGY have formal semantics, we are able to provide proofs of correctness for the different algorithms implementation.

Related Work. Several definitions of graph rewriting are available, using different kinds of graphs and rewrite rules (see, for instance, [18, 19, 20, 21, 22, 23, 24, 25]). To model social networks in this paper, we define *directed port graphs*

with attributes and associated notions of strategic rewriting, based on the general notion of *port graph rewriting* presented in [13]. An alternative solution using undirected edges and nodes with two ports called "In" and "Out" to simulate edge direction was previously developed and successfully applied in [8], however, having oriented edges as a primitive concept makes it much easier to represent social relationships that are naturally asymmetric. To facilitate the specification of network algorithms, we have also extended the concept of port graph rewrite rule to incorporate conditional matching, reminiscent of similar solutions found in ELAN [26] and GP [27].

Although many data sets, extracted from various real-world social networks, are publicly available[1], in order to test new ideas, demonstrate the generality of a new technique, or design and experiment with stochastic algorithms on a sufficiently large sample of network topologies, it is sometimes convenient to use randomly generated networks as they can be fine-tuned to produce graphs with specific properties (number of nodes, edge density, edge distribution...). Several generative models of random networks are available (e.g., [28, 29, 30, 31, 32]). Some, like the Erdös–Rényi (ER) model [28], do not guarantee any specific property regarding their final topology, whereas others can show small-world characteristics [29], distinctive (scale-free) edge distributions [30] or both at the same time [31]. In this paper, we show how to generate such models using labelled port graphs, rules and strategies. Moreover, we take advantage of the visual and statistical features available in PORGY to tune the algorithms: our experimental results guide and validate the parameter choices made in the generation algorithms, ensuring the generated networks satisfy the required properties.

Afterwards, we implement four propagation algorithms using the labelled graphs, rules and strategies concepts mentioned above. While three of them are based on existing models described in previous work [14, 15, 16], the fourth model is an original creation, specifically built to incorporate the privacy-preserving features described in [16] to the propagation mechanism proposed in [15].

[1] For instance from `http://snap.stanford.edu` or `http://konect.uni-koblenz.de/`

The paper is organised as follows. Section 2 introduces the modelling concepts: labelled port graphs, rewriting, derivation tree, strategy and strategic graph programs. We develop social network generation algorithms in Section 3, and propagation algorithms in Section 4. Section 5 briefly describes a framework for designing and experimenting with social network models. In Section 6, we conclude and give directions for future work.

2. Labelled Graph Rewriting for Social Networks

A social network [33] is usually described as a graph where the nodes represent the users and the edges represent their relationships. Some real-world social relations involve mutual recognition (e.g., friendship), whereas others present an asymmetric model of acknowledgement (e.g., Twitter, where one of the users is a *follower* while the other is a *followee*). It is thus natural to represent such relations using directed graphs. In this paper, we model social networks using labelled directed port graphs, as defined below.

2.1. Directed Port Graphs for Social Networks

Roughly speaking, a port graph is a graph where edges are attached to nodes at specific points, called *ports*. Nodes, ports and edges are labelled using records over a *signature* ∇ comprising pairwise disjoint sets $\nabla_{\mathscr{A}}$ of attributes, $\mathcal{X}_{\mathscr{A}}$ of attribute variables, $\nabla_{\mathscr{V}}$ of values and $\mathcal{X}_{\mathscr{V}}$ of value variables. We assume $\nabla_{\mathscr{A}}$ contains distinguished elements *Name*, $(In/Out)Arity$, *Connect*, *Attach*, and *Interface*. Values in $\nabla_{\mathscr{V}}$ are assumed to be of basic data types such as *strings*, *integers*, *Booleans*,... or to be well-typed computable expressions built using ∇ and basic types.

A record r over ∇ is a set of pairs *(attribute, value)*: $\{(a_1, v_1), \ldots, (a_n, v_n)\}$, where for $1 \leq i \leq n$, $a_i \in \nabla_{\mathscr{A}} \cup \mathcal{X}_{\mathscr{A}}$, each a_i occurs only once in r, and there is one distinguished attribute *Name*, and $v_i \in \nabla_{\mathscr{V}}$ for $1 \leq i \leq n$.

The function *Atts* returns all the attributes in a given record: $Atts(r) = \{a_1, \ldots, a_n\}$ if $r = \{(a_1, v_1), \ldots, (a_n, v_n)\}$. As usual, $r.a_i$ denotes the value v_i of the attribute a_i in r.

The attribute *Name* identifies the record in the following sense: for all r_1, r_2, $Atts(r_1) = Atts(r_2)$ if $r_1.Name = r_2.Name$.

Definition 1 (Directed port graph). *Given sets $\mathscr{N}$, $\mathscr{P}$, $\mathscr{E}$ of nodes, ports and edges, a* directed port graph *(or simply port graph, since there will be no ambiguity) G over a* signature ∇ *is a tuple $(N, P, E, \mathcal{L})$ where*

- *$N \subseteq \mathscr{N}$ is a finite set of nodes; $n, n', n_1, \ldots$ range over nodes.*
- *$P \subseteq \mathscr{P}$ is a finite set of ports; $p, p', p_1, \ldots$ range over ports.*
- *$E \subseteq \mathscr{E}$ is a finite set of edges between ports; $e, e', e_1, \ldots$ range over edges. Edges are directed and two ports may be connected by more than one edge.*
- *$\mathcal{L}$ is a labelling function that returns, for each element in $N \cup P \cup E$, a record such that:*
  - *for each edge $e \in E$, $\mathcal{L}(e)$ contains an attribute Connect whose value is the ordered pair (p_1, p_2) of ports connected by e.*
  - *for each port $p \in P$, $\mathcal{L}(p)$ contains an attribute Attach whose value is the node n which the port belongs to, and two attributes, InArity and OutArity, whose values are the number of incoming and outgoing edges connected to this port, respectively.*
  - *For each node $n \in N$, $\mathcal{L}(n)$ contains an attribute Interface whose value is the set of names of ports in the node:*

$$\{\mathcal{L}(p_i).Name \mid \mathcal{L}(p_i).Attach = n\}$$

We assume that $\mathcal{L}$ satisfies the following constraint:

$$\mathcal{L}(n_1).Name = \mathcal{L}(n_2).Name \Rightarrow \mathcal{L}(n_1).Interface = \mathcal{L}(n_2).Interface.$$

We present in Figure 1 an example of a labelled directed port graph. In this example, nodes have attributes *State*, *Marked*, and *Tau* (used in the algorithms given in the next sections), as well as an attribute *Colour*, for visual purposes.

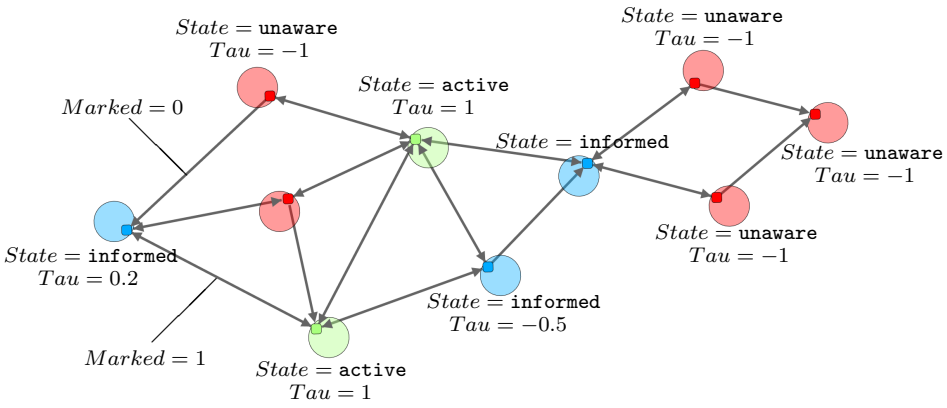


Figure 1: Example of port graph for a toy social network

According to definition 1, nodes with the same name (i.e., the same value for their attribute *Name*) have the same set of port names (i.e., the same interface), with similar attributes but with possibly different values. Variables, however, may be used to denote any value.

If an edge $e \in E$ goes from n to n', we say that n' is *adjacent* to n (not conversely) and thus that n' is a neighbour of n. The set of nodes adjacent to a subgraph F in G consists of all the nodes in G *outside* F and adjacent to any node in F. $Ngb(n)$ is used to denote the set of neighbours of the node n.

The advantage of using port graphs rather than plain graphs is that they allow us to express in a more structured and explicit way the properties of the connections, since ports represent the connecting points between edges and nodes. However, the counterpart is that the implementation, rules and matching operations are more complex. So, whenever possible, it is simpler and more efficient to keep the number of ports for each node to a minimum.

In the social network models we use in this paper, nodes representing users have only one port gathering directed connections. While this is sufficient in many cases, when dealing with real social networks, multiple ports are useful, either to connect users according to the nature of their relation (e.g., friend, parent, co-worker . . .) or to model situations where a user is connected to friends via different social networks. The full power of port graphs is indeed necessary in

multi-layer networks [34] where edges are assigned to different layers and where nodes are shared. In that case, different ports are related to different layers, which can improve modularity of design, readability and matching efficiency through various heuristics. This is however a topic left for future work.

2.2. Located Rewriting

A *port graph rewrite rule* $L \Rightarrow R$ is itself a port graph, consisting of two subgraphs L and R and an *arrow* node that encodes in particular the correspondence between ports in L and ports in R.

Definition 2 (Port graph rewrite rule). *A port graph rewrite rule is a port graph consisting of:*

- *two directed port graphs L and R over the signature ∇, respectively called* left-hand side *and* right-hand side*, such that all the variables in R occur in L, and R may contain records with expressions;*
- *an* arrow *node with a set of edges that each connect a port of the arrow node to a port in L and a port in R. The arrow node has for $Name$ $\Rightarrow$. It also has an optional attribute* Where *whose value is a Boolean expression involving elements of L (edges, nodes, ports and their attributes).*

The edges connecting L and R to the arrow node are used to rewire the graph when the rule is applied. See Figure 2 (top right-hand side corner) for an example of port graph rewrite rule.

When modelling rumour propagation, the rules never suppress nor add new nodes. Moreover, since there is only one port per node, there is no ambiguity on the rewiring between left and right-hand sides. In that case, the structure and visualisation of the arrow node is much simpler. However, this only holds when the network's structure does not change.

The introduction of the *Where* attribute is inspired from the GP programming system [27] (and from a more general definition given in ELAN [26]), in which a rule may have a condition introduced by the keyword `where`. The

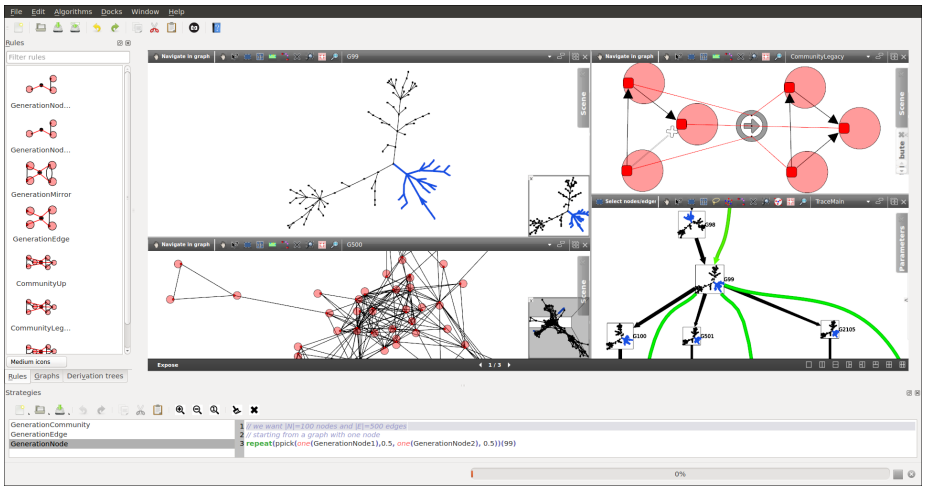


Figure 2: A screenshot of PORGY in action.

Boolean expression introduced with the keyword `where` is in this paper used to specify the absence of certain edges. For instance, a condition `where not Edge(n,n')` requires that no edge exists between the nodes n and n'. This condition is checked at matching time.

Let us now recall the notion of port graph morphism [13]. Let G and H be two port graphs over the same signature ∇. A *port graph morphism* $f : G \mapsto H$ maps nodes, ports and edges of G to those of H such that the attachment of ports and the edge connections are preserved, and all attributes and values are preserved except for variables in G, which must be instantiated in H. Variables are useful to specify rules where some attributes of the left-hand side are not relevant for the application of the transformation. Intuitively, the morphism identifies a subgraph of H that is equal to G except at positions where G has variables (at those positions, H could have any instance).

Definition 3 (Match). *Let $L \Rightarrow R$ be a port graph rewrite rule and G a port graph. We say a* match *$g(L)$ of the left-hand side (also called a* redex*) is found if: there is an injective port graph morphism g from L to G (hence $g(L)$ is a subgraph of G) such that if the arrow node has an attribute* Where *with value C, then C is true of $g(L)$, and for each port p in L that is not connected to the*

arrow node, $g(p)$ is not *connected with* $G - g(L)$.

This last point ensures that ports in L that are not connected to the arrow node are mapped to ports in $g(L)$ that have no edges connecting them with ports outside the redex, to avoid dangling edges in rewriting steps.

Several injective morphisms g from L to G may exist (leading to different rewriting steps); they are computed as solutions of a *matching* problem from L to (a subgraph of) G.

Definition 4 (Rewriting step). *A* rewriting step *on* G *using a rule* $L \Rightarrow R$ *and a match* $g(L)$*, written* $G \rightarrow^{g}_{L \Rightarrow R} G'$*, transforms* G *into a new graph* G' *obtained from* G *in three steps:*

- *In the* build *step, a copy* $R_c = g(R)$ *(i.e., an instantiated copy of the port graph* R*) is added to* G*.*
- *The* rewiring *phase then redirects edges from* G *to* R_c *as follows:*

  For each port p *in the arrow node:*

  If $p_L \in L$ *is connected to* p*:*

  for each port $p^i_R \in R$ *connected to* p*,*

  find all the ports p^k_G *in* G *that are connected to* $g(p_L)$ *and are not in* $g(L)$*, and redirect each edge connecting* p^k_G *and* $g(p_L)$ *to connect* p^k_G *and* $p^i_{R_c}$*.*
- *The* deletion *phase simply deletes* $g(L)$*. This creates the final graph* G'*.*

Given a finite set $\mathcal{R}$ of rules, a port graph G *rewrites* to G', denoted by $G \rightarrow_{\mathcal{R}} G'$, if there is a rule r in $\mathcal{R}$ and a morphism g such that $G \rightarrow^{g}_{r} G'$. This induces a reflexive and transitive relation on port graphs, called *the rewriting relation*, denoted by $\rightarrow^{*}_{\mathcal{R}}$. A port graph on which no rule is applicable is *irreducible*.

To facilitate the specification of graph transformations, we use the concept of *located graph* from [13].

Definition 5 (Located graph). *A* located graph G^Q_P *consists of a port graph* G *and two distinguished subgraphs* P *and* Q *of* G*, called respectively the* position subgraph*, or simply* position*, and the* banned subgraph*.*

In a located graph G_P^Q, P is the subgraph of G under study (the focus of the transformations), and Q is a protected subgraph, where transformations are forbidden.

When applying a port graph rewrite rule, not only the underlying graph G but also the position and banned subgraphs may change. A *located rewrite rule*, defined below, specifies two disjoint subgraphs M and M' of the right-hand side R that are respectively used to update the position and banned subgraphs. If M (resp. M') is not specified, R (resp. the empty graph $\emptyset$) is used as default. Below, we use the operators $\cup, \cap, \setminus$ to denote union, intersection and complement of port graphs. These operators are defined in the natural way on port graphs considered as sets of nodes, ports and edges.

Definition 6 (Located rewrite rule). *A* located rewrite rule *is given by a port graph rewrite rule $L \Rightarrow R$, with two disjoint subgraphs M and M' of R and optionally, a subgraph W of L. It is denoted $L_W \Rightarrow R_M^{M'}$.*

We write $G_P^Q \rightarrow^g_{L_W \Rightarrow R_M^{M'}} {G'}_{P'}^{Q'}$ and say that the located graph G_P^Q rewrites to *${G'}_{P'}^{Q'}$* using *$L_W \Rightarrow R_M^{M'}$* at position *P* avoiding *Q, if $G \rightarrow_{L \Rightarrow R} G'$ with a morphism g such that $g(L) \cap P = g(W)$ or simply $g(L) \cap P \neq \emptyset$ if W is not provided, and $g(L) \cap Q = \emptyset$. The new position subgraph P' and banned subgraph Q' are defined as $P' = (P \setminus g(L)) \cup g(M)$, $Q' = Q \cup g(M')$; if M (resp. M') are not provided then we assume $M = R$ (resp. $M' = \emptyset$).*

Example 7. *In influence propagation, carefully managed position and banned subgraphs are used to avoid several consecutive activations of the same neighbours. Another usage is to select a specific community in the social network where the propagation should take place.*

In general, for a given located rule $L_W \Rightarrow R_M^{M'}$ and located graph G_P^Q, several rewriting steps at P avoiding Q may be possible. Thus, the application of the rule at P avoiding Q may produce several located graphs. A *derivation*, or computation, is a sequence of rewriting steps. If all the derivations are finite, the system is said to be *terminating*. A *derivation tree* from G is made of all

possible computations (including possibly infinite ones). See Figure 2 (bottom right-hand side corner) for an example of derivation tree. *Strategies* are used to specify the rewriting steps of interest, by selecting branches in the derivation tree.

2.3. Strategic graph programs

A *strategic graph program* consists of a *located graph*, a set of located rewrite rules, and a strategy expression that combines applications of located rules and focusing constructs. In the context of social networks, the focusing primitives are convenient to restrict the application of rules to specific parts of the graph. Subgraphs of a given graph can be defined by specifying simple properties, expressed with attributes of nodes, edges and ports.

The most basic strategies are id, which always succeeds, and fail, which always fails. More interesting strategies can be built by composition; a full description of the strategy language can be found in [13]. We briefly explain below the constructs used in this paper.

The primary construct is a located rule, which can only be applied to a located graph G_P^Q if at least a part of the redex is in P, and does not involve Q. When probabilities $\pi_1, \ldots, \pi_k \in [0, 1]$ are associated to rules $T_1, \ldots, T_k$ such that $\pi_1 + \cdots + \pi_k = 1$, the strategy $\texttt{ppick}(T_1, \pi_1, \ldots, T_k, \pi_k)$ picks one of the rules for application, according to the given probabilities.

$\texttt{all}(T)$ denotes all possible applications of the transformation T on the located graph at the current position, creating a new located graph for each application. In the derivation tree, this creates as many children as there are possible applications.

$\texttt{one}(T)$ computes only one of the possible applications of the transformation and ignores the others; more precisely, it makes an equiprobable choice between all possible applications.

Similar constructs exist for focusing expressions, which are used to define positions for rewriting in a graph, or to define positions where rewriting is not allowed: $\texttt{one}(F)$ returns one node in the subgraph defined by F and $\texttt{all}(F)$

returns the full F. When not specified, F stands for $\texttt{all}(F)$. In this paper, F is defined using the following elements:

- `crtGraph`, `crtPos` and `crtBan`: applied to a located graph G_P^Q, return respectively the whole graph G, the position subgraph P and the banned subgraph Q.

- $\texttt{property}(F, \rho)$ is used to select elements of a given graph that satisfy a certain property, specified by ρ. It can be seen as a filtering construct: if the expression F generates a subgraph G' then $\texttt{property}(F, \rho)$ returns only the nodes and/or edges from G that satisfy the decidable property $\rho = Elem, Expr$. Depending on the value of $Elem$, the property is evaluated on nodes, ports, or edges.

- $\texttt{ngb}(F, \rho)$ returns a subset of the neighbours (i.e., adjacent nodes) of F according to ρ. Note that the edge direction is taken into account, to emphasise it, we also introduce $\texttt{ngbOut}(F, \rho)$ and its counterpart $\texttt{ngbIn}(F, \rho)$. If `edge` is used, i.e., if we write $\texttt{ngb}(F, \texttt{edge}, Expr)$, it returns all the neighbours of F connected to F via edges which satisfy the expression $Expr$.

- $\texttt{setPos}(D)$ (resp. $\texttt{setBan}(D)$) sets the position subgraph P (resp. Q) to be the graph resulting from the expression D. This operation always succeeds (i.e., returns `id`).

- $\texttt{isEmpty}(F)$ returns `id` if F denotes the empty graph and `fail` otherwise. It is defined as $F = \emptyset$. This last focusing operation, although derivable, facilitates the implementation.

The following constructs are used to build strategies.

- $S;S'$ represents sequential application of S followed by S'.

- $\texttt{if}(S)\texttt{then}(S')\texttt{else}(S'')$ checks if the application of S on (a copy of) G_P^Q returns `id`, in which case S' is applied to (the original) G_P^Q, otherwise S'' is applied to the original G_P^Q. The $\texttt{else}(S'')$ part is optional.

- `repeat`(S)[max n] simply iterates the application of S until it fails; if max n is specified, then the number of repetitions cannot exceed n.

- (S)`orelse`(S') applies S if possible, otherwise applies S'. It fails if both S and S' fail.

- `try`(S) behaves like S if S succeeds, but will returns id if S fails. It is a derived operation which is defined as (S)`orelse`(id).

- When probabilities $\pi_1, \ldots, \pi_k \in [0, 1]$ are associated to strategies $S_1, \ldots, S_k$ such that $\pi_1 + \cdots + \pi_k = 1$, the strategy `ppick`$(S_1, \pi_1, \ldots, S_k, \pi_k)$ picks one of the strategies for application, according to the given probabilities. This construct generalises the probabilistic constructs on rules and positions.

Probabilistic features of the strategy language, through the use of the `ppick`() construct, are illustrated in Section 3 for social network generation. The propagation models in Section 4 illustrate how record expressions are used to compute attribute values and how these are updated through application of rules.

A more complete definition of strategic graph programs and their semantics can be found in [13], where it is proved that the derivation tree in which each rewrite step is performed according to the strategy –let us call it the *strategic derivation tree*– is actually a subtree of the derivation tree of the rewrite system without strategy. The strategic derivation tree is a valuable concept because it records the history of the transformations and provides access to generated models. It is, by itself, a source of challenging questions, such as detecting isomorphic models and folding the tree, finding equivalent paths and defining the "best ones", abstracting a sequence of steps by a composition strategy, or managing the complexity of the tree and its visualisation.

3. Social network generation

In this section we address the problem of generating graphs with a *small-world property* as defined in [29]. Such graphs are characterised by a small diameter –the average distance between any pair of nodes is short– and strong

local clustering –for any pair of connected nodes, both tend to be connected to the same neighbouring nodes, thus creating densely linked groups of nodes called *communities*. Popularised by Milgram in [35], small-world graphs are a perfect case study for information propagation in social networks due to their small diameter allowing a quick and efficient spreading of information.

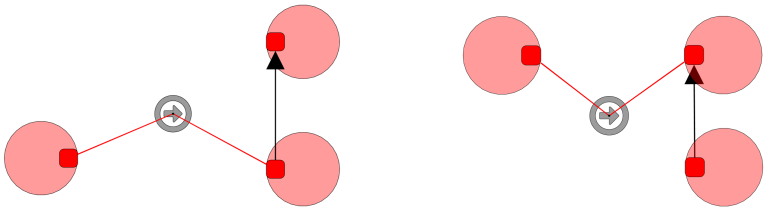
Our goal is to design an algorithm to generate small-world graphs of a given size, that is, for which the number of nodes $|N|$ and directed edges $|E|$ are given *a priori*. Moreover, the graphs generated should satisfy the following conditions: they must have only one connected component, thus $|E| \geq |N| - 1$; they should be simple, that is, any ordered pair of nodes (n, n') can only be linked once, thus the maximum number of edges is $|E|_{max} = |N| \times (|N| - 1)$; finally, the number of communities should be randomly decided during the generation process.

A few previous works have explored the idea of using rules to generate networks. In [36], the authors define and study probabilistic inductive classes of graphs generated by rules which model spread of knowledge, dynamics of acquaintanceship and emergence of communities. Below we present a new algorithm for social network generation that follows a similar approach, however, we have adjusted its generative rules to cope with directed edges and ensure the creation of a graph with a single connected component. This is achieved by performing the generation through local additive transformations, each only creating new elements connected to the sole component, thus increasingly making the graph larger, more intricate and more interesting to study.

Starting from one node, the generation is divided into three phases imitating the process followed by real-world social networks. Whenever new users first join the social network, their number of connections is very limited, mostly to the other users who have introduced them to the social network (Sect. 3.1). During the second phase, these new users can reach to the people they already know personally, thus creating new connections within the network, which may seem random for any spectator only aware of the present social network (Sect. 3.2). Finally, the users get to know the people with whom they are sharing friends in the network, potentially leading to the creation of new connections (Sect. 3.3).

3.1. Generation of a directed acyclic port graph

The first step toward the construction of a directed port graph $G = (N, E, P, \mathcal{L})$ uses the two rules shown in Figures 3(a) and 3(b). Both rules apply to a single node and generate two linked nodes (thus each application increases by one the number of nodes and also the number of edges). The difference between these two rules lies in the edge orientation as Rule 3(a) creates an outgoing edge on the initiating node, while Rule 3(b) creates an incoming edge.



(a) Rule *GenerationNode1*. (b) Rule *GenerationNode2*.

Figure 3: Rules used for generating and re-attaching nodes to pre-existing node with a directed edge going from the pre-existing node to the newly added node in (a) or oriented in the opposite direction in (b).

Strategy 1: *Node generation*: Creating a directed acyclic graph of size N

```
1 //equiprobabilistic application of the two rules used for generating nodes
2 repeat(
3   ppick(one(GenerationNode1), 0.5,
4         one(GenerationNode2), 0.5)
5 )(|N| − 1) // Generation of N nodes
```

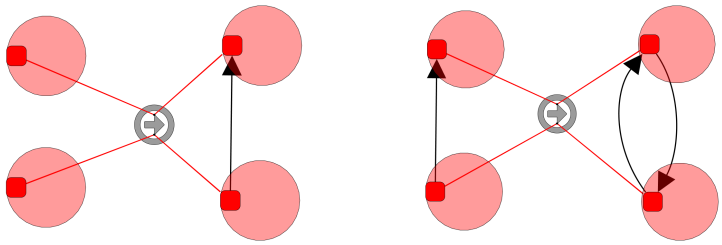
The whole node generation is achieved during this first phase and managed using Strategy 1. It repeatedly applies the generative rules $|N|-1$ times so that the graph reaches the appropriate number of nodes. As mentioned earlier, each rule application also generates a new edge, which means that once executed, Strategy 1 produces a graph with exactly $|N|$ nodes and $|N|-1$ edges. The orientation of each edge varies depending of the rule applied (either 3(a) or 3(b)),

moreover, their application using the `ppick()` construct ensures an equiprobable choice between the two rules.

3.2. Creating complementary connections

During this phase, we either create seemingly random connections between the network users or reciprocate already existing single-sided connections.

We use two rules to link existing nodes, thus creating a new additional edge with each application. The first rule (Fig. 4(a)) simply considers two nodes and adds an edge between them to emulate the creation of a (one-sided) connection between two users. The second rule (Fig. 4(b)) reciprocates an existing connection between a pair of users: for two nodes $n, n' \in N$ connected with an oriented edge (n', n), a new oriented edge (n, n') is created; it is used to represent the mutual appreciation of users in the social network. Note that, because each node is randomly chosen among the possible matches, we do not need to create alternative versions of these rules with reversed oriented edges.



(a) Rule *GenerationEdge*. (b) Rule *GenerationMirror*.

Figure 4: Rules generating additional connections: (a) between two previously unrelated nodes, (b) by reciprocating a pre-existing connection.

In both rules, the existence of edges between the nodes on which the rule applies should be taken into account: the rules should not create an edge if a similar one already exists (since we aim at creating a simple graph rather than a multi-graph). This can be achieved by adding a condition "`where not Edge(n,n')`" (see Definition 2), or by using position constructs to restrict the elements to be considered during matching. We use the latter solution here.

In Strategy 2, we first filter the elements to consider during the matching. We randomly select one node among the nodes whose outgoing arity ($OutArity$) is lower than the maximal possible value (i.e., $|N|-1$), and we ban all its outgoing neighbours as they cannot be considered as potential matching elements. Then, Rule 4(a) or Rule 4(b) are equiprobably applied to add a new edge from the selected node. By banning neighbours, we ensure that future applications of the rule will not use those nodes, that is, the rule will only apply on pairs of nodes not already connected. This ensures that the graph is kept simple (i.e., only one edge per direction between two nodes).

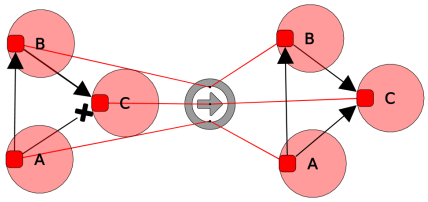
Strategy 2: *Edge generation*: addition of $|E'|$ edges if possible.

```
repeat(
//select one node with an appropriate number of neighbours
  setPos(one(property(crtGraph, node, OutArity < |N| − 1)));
//for this node, forbid rule applications on its outgoing neighbours
  setBan(all(ngbOut(crtPos, node, true)));
//equiprobable application of the edge generation rules
  ppick((one(GenerationEdge))orelse(one(GenerationMirror)), 0.5,
          (one(GenerationMirror))orelse(one(GenerationEdge)), 0.5);
)(|E′|)
```

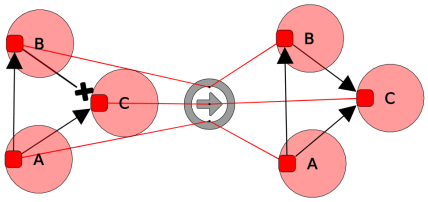
In this phase, we create $|E'|$ edges, where $|E'| < (|E| - |N| + 1)$ to keep the number of edges below $|E|$. The use of the `orelse` construct allows testing all possible rule application combinations, thus, if one of the rules can be applied, it is found. If no rule can be applied, the maximum number of edges in the graph has been reached, i.e., the graph is complete. If the value of $|E'|$ is not too high, we are left with $(|E| - |E'| - |N| + 1)$ remaining edges to create in the next step for enforcing communities within G.

3.3. Construction of communities

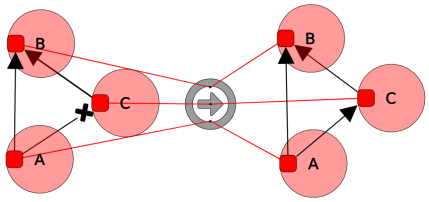
To create a realistic social network, we now add communities. For this, we need to ensure that the links between users follow certain patterns. Based on ideas advanced in several previous works (e.g., [37, 36, 38, 39]), we focus on triad configurations (i.e., groups formed by three users linked together). Our community generation algorithm uses three rewrite rules, introduced in Figure 5.



(a) Rule *CommunityLegacy*.



(b) Rule *CommunityDown*.



(c) Rule *CommunityUp*.

Figure 5: Generation of additional connections based on triads. Two distinctive edge types are used: standard arrow edges for representing existing connections and cross-shaped headed edges for indicating edges which should not exist during the matching phase.

The first triad rule (Fig. 5(a)) considers how a first user (A) influences a second user (B) who influences in turn a third user (C)[2]. The second rule (Fig. 5(b)) shows two users (B and C) being influenced by a third user (A)[3]. The last rule (Fig. 5(c)) depicts one user (B) being influenced by two other

[2]This situation can produce some sort of transitivity as "the idol of my idol is my idol", meaning that A is much likely to influence C. We use here the term "idol" instead of the more classical "friend" because we only consider single-sided relations.

[3]When in this position, the users B and C might start exchanging (similar connections, common interests...), thus creating a link between the two of them.

users (A and C)[4].

The three rules use a `where not Edge(n,n')` condition to forbid the existence of an edge between two matching nodes.

Strategy 3 is used to drive the three rules. Like the previous strategy, this one aims at equiprobably testing all possible rule combinations.

Strategy 3: *Community generation*: creating edges to strengthen communities

```
1  repeat(
2   ppick(
3     (one(CommunityDown))orelse(
4       ppick(
5         (one(CommunityUp))orelse(one(CommunityLegacy)), 0.5,
6         (one(CommunityLegacy))orelse(one(CommunityUp)), 0.5)
7       ), 1/3,
8     (one(CommunityUp))orelse(
9       ppick(
10        (one(CommunityLegacy))orelse(one(CommunityDown)), 0.5,
11        (one(CommunityDown))orelse(one(CommunityLegacy)), 0.5)
12      ), 1/3,
13    (one(CommunityLegacy))orelse(
14      ppick(
15        (one(CommunityDown))orelse(one(CommunityUp)), 0.5,
16        (one(CommunityUp))orelse(one(CommunityDown)), 0.5)
17      ), 1/3)
18 )(|E| − |E'| − |N| + 1)
```

[4]This case can happen when A and C are well-versed about a common subject of interest which is of importance to B. A link is thus created between the two influential users.

3.4. Resulting network generation

For the sake of simplicity, the strategies presented above make equiprobable choices between rules. The probabilities may of course be modified to take into account specific conditions present in the modelled system. Whatever the chosen probabilities are, the following result holds.

Proposition 8. *Given three positive integer parameters $|N|$, $|E|$, $|E'|$, such that $|N| - 1 \leq |E| \leq |N| \times (|N| - 1)$ and $|E'| \leq |E| - |N| + 1$, let the strategy $S_{|N|,|E|,|E'|}$ be the sequential composition of the strategies* Node generation*,* Edge generation *and* Community generation *described above, and G_0 be a port graph composed of one node with one port. The strategic graph program $[S, G_0]$ terminates with a simple and weakly-connected directed port graph G with $|N|$ nodes and $|E|$ edges.*

Proof 9. *The termination property is a consequence of the fact that the three composed strategies have only one command which could generate an infinite derivation (the repeat loop) but in the three cases, there is a limit on the number of iterations (i.e., it is a bounded repeat).*

Since the program terminates, we can use induction on the number of rewriting steps to prove that the generated port graphs are directed, simple (at most one edge in each direction between any two nodes) and weakly connected (connected when direction of edges is ignored). This is trivially true for G_0 and each rewrite step preserves these three properties, thanks to the positioning strategy that controls the out degree in Edge generation *(Strategy 2) and the forbidden edges in the rules for* Community generation *(Figure 5). As the strategic program never fails, since a repeat strategy cannot fail, this means that a finite number of rules has been applied and the three properties hold by induction.*

It remains to prove that the number of nodes and edges is as stated. Observe that by construction, the strategy Node generation *creates a new node and a new edge at each step of the repeat loop, exactly $|N| - 1$, and is the only strategy that creates new nodes. Hence, after applying the* Node generation *strategy, the graph created has exactly $|N|$ nodes and $|N| - 1$ edges. The strategies* Edge

generation *and* Community generation *create a new edge at each step of the repeat loop, so respectively* $|E'|$ *and* $|E| - |E'| - |N| + 1$. *As a result, when the strategy* S *terminates, the number of edges created is equal to* $\big(|N|-1\big)+\big(|E'|\big)+\big(|E| - |E'| - |N| + 1\big) = |E|$. □

The method presented above can easily be extended to create graphs with more than one component. One has to use a number of starting nodes equal to the number of desired connected components and ensure that no edge is created between nodes from different components. The generative rules and strategies can then be applied on each component iteratively or in parallel (parallel application of rules is possible but beyond the scope of this paper).

3.5. Implementation and Experimental Validation

We use the PORGY system [17] to experiment with our generative model. The latest version of the rewriting platform[5] is available either as source code or binaries for MacOS and Windows machines.

Figures 6 and 7 are two examples of social networks generated using a sequential composition of the previous strategies. Although both graphs have the same number of nodes and edges ($|N| = 100$ and $|E| = 500$), they have been generated with different $|E'|$, respectively $|E'| = 50$ for Fig. 6 and $|E'| = 0$ for Fig. 7. This changes the number of purely random edges created in the resulting graph and explains why the first graph seems to visually present less structure than the other one. Conversely, a graph with only randomly assigned edges could be generated with $|E'| = |E| - |N| + 1$.

To ensure that our constructions present characteristics of real-world social networks, we have performed several generations using different parameters and measured the *characteristic path length* –the average number of edges in the shortest path between any two nodes in the graph– and the *clustering coefficient* –how many neighbours of a node n are also connected with each other– as defined in [29]. In a typical random graph, e.g., a graph generated using the

[5] PORGY website: `http://porgy.labri.fr`

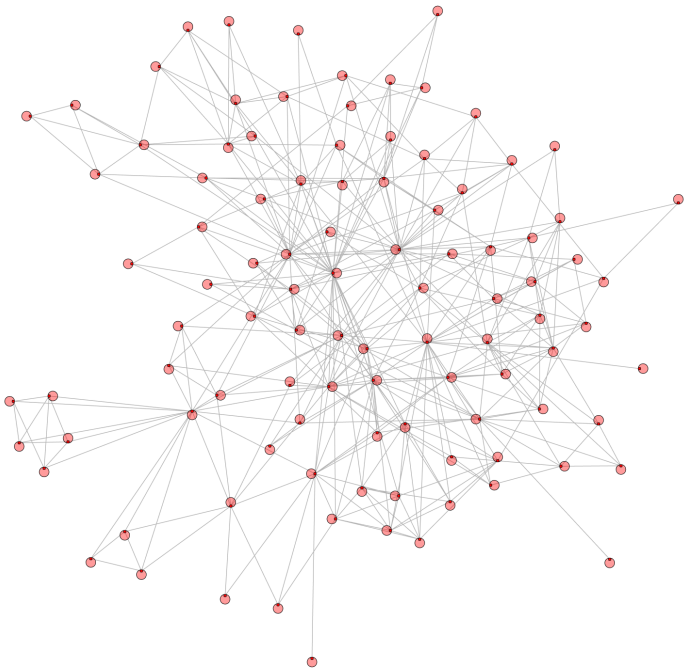


Figure 6: A generated social network. $|N| = 100$ nodes, $|E| = 500$ edges and $|E'| = 50$. With these parameters, the average characteristic path length is $L \simeq 2.563$ and the average clustering coefficient is $C \simeq 0.426$.

Erdös–Rényi model [28] or using our method with the parameters $|N| = 100$ nodes, $|E| = 500$ edges and $|E'| = |E| - |N| + 1 = 401$, the average characteristic path length is very short ($L \simeq 2.274$), allowing information to go quickly from one node to another, but the clustering coefficient is low ($C \simeq 0.101$), implying the lack of well-developed communities. However, with the parameters used in Figure 6 (respectively, Figure 7), we retain a short characteristic path length $L \simeq 2.563$ (resp. $L \simeq 3.372$) while increasing the clustering coefficient $C \simeq 0.426$ (resp. $C \simeq 0.596$), thus matching the characteristics of small-world graphs: a small diameter and strong local clustering.

The graphs generated using our method can be subsequently used as any randomly generated network. For instance, we have used such graphs in [8] to study the evolution of different information propagation models. In the following

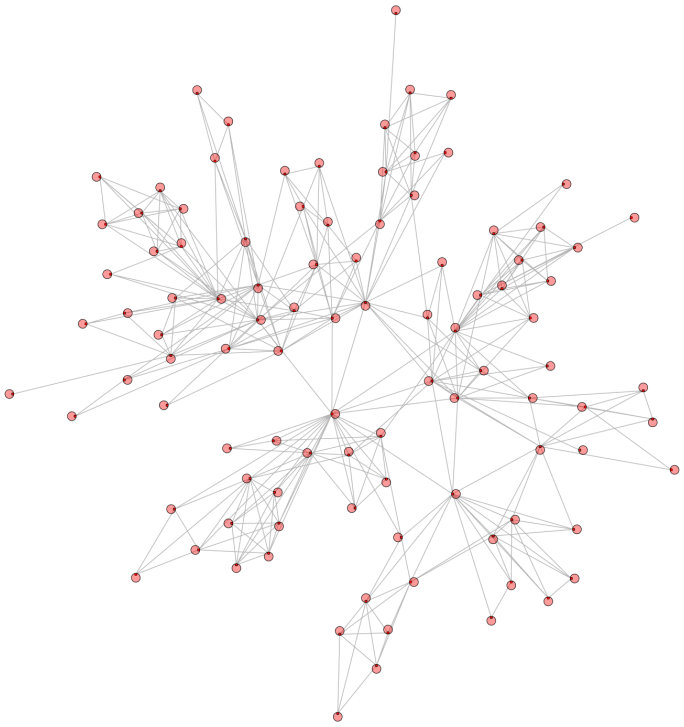


Figure 7: A generated social network. $|N| = 100$ nodes, $|E| = 500$ edges and $|E'| = 0$. With these parameters, the average characteristic path length is $L \simeq 3.372$ and the average clustering coefficient is $C \simeq 0.596$.

section, we recall these propagation models and introduce their translation as strategic graph programs using directed located port graphs, located rewrite rules and strategies.

4. Propagation in social networks

In social networks, propagation occurs when users perform a specific action (such as relaying information, announcing an event, spreading gossip, sharing a video clip), thus becoming *active*. Their neighbours are then informed of their state change, and are offered the possibility to become active themselves if they perform the same action. The process then reiterates as newly active neighbours share the information with their own neighbours, propagating the

activation from peer to peer throughout the whole network.

To replicate this phenomena, some propagation models opt for entirely probabilistic activations (e.g., [14, 40, 41]), where the presence of only one active neighbour is often enough to allow the propagation to occur, while others (e.g., [15, 42, 43]) use threshold values, building up during the propagation. Such values represent the *influence* of one user on his neighbours or the tolerance towards performing a given action (the more solicited a user is, the more inclined he becomes to either activate or utterly resist). In general, several propagations may happen in one network at the same time, but most propagation models focus only on one action (e.g., relaying a specific information) as the other propagations are likely to be about entirely different subjects, thus creating few if any interference.

In [8], two basic propagation models were specified using strategic rewriting: the *independent cascade* model **IC** [14] and the *linear threshold* model **LT** [15]. In this section, we first recall the definition of these two models and show in which manner they can be described as instances of the same strategic graph program. We then consider another propagation model, called *Riposte* (**RP**), described in [16]. For this model, it is shown that an information deemed interesting by a sufficiently large fraction of the population is more likely to spread widely, whereas an information that only a few people consider interesting will not spread far beyond the set of users initially exposed to it. Moreover, when observing the information propagation process (more precisely, the users' re-posts), one cannot determine with sufficient confidence the opinion of any single user concerning the information that is disseminated. We implement the algorithm as a strategic graph program. Finally, we use the rules and strategies modularity to develop a new propagation model combining features of **LT** and **RP**, which we also implement as a strategic graph program.

In the following implementations as strategic graph programs of the various propagation models, several common ingredients are used:

- we assume that, at any given time, each node is in a precise *state*, which determines its involvement in the current spreading of information. States are

represented by one of the following values: *unaware* for those who have not (yet) heard of the action, *informed* to describe those who have been informed of the action/influenced by their neighbours, or *active* to qualify those who have been successfully influenced and decided to propagate the action themselves. We encode this information on each node using an attribute *State*, which can take one of these three values as a string of characters: `unaware`, `informed`, or `active`. For visualisation purposes, an attribute *Colour* is associated to *State* to colour the nodes in `red`, `blue`, or `green`, respectively.

- the rules we use to express the models describe how the nodes' states evolve. An `unaware` node becomes `informed` when at least one of its `active` neighbours tries to influence it, and an `informed` node becomes `active` when its influence level is sufficiently high. These two distinct steps correspond to the two basic *State* transformations we need to represent using the rewrite rules. We name the first step the *influence trial*, during which an `active` node n tries to influence an inactive neighbour n' (where n' is either `unaware` or just `informed`). The following step is the *activation* of n', where the node becomes `active` once it has been successfully influenced.

- for each model, we use an attribute called *Tau* to store the influence level of the `informed` nodes. Computed/updated during the *Influence trial* step, this attribute is by default initialised to -1 and can take a numerical value in $[-1, 1]$.

These principles are carried out in the following examples of propagation mechanisms. With each model, we introduce visual representations of the rules applied to perform the rewriting operations. We mention in their left-hand sides the attributes that are used in the matching process, and in their right-hand sides the attributes whose values are modified during the rewriting step. The specifications are detailed hereafter.

*4.1. The independent cascade model (**IC**)*

We first describe a basic form of the **IC** model as introduced in [14]. This model has several variations (*e.g.* [44, 43]) allowing, for instance, to simulate the propagation of diverging opinions in a social network [40].

Definition 10 (Propagation process in IC). *For each pair of adjacent nodes (n, n'), let $p_{n,n'}$ be the influence probability from n on n' ($0 \leq p_{n,n'} \leq 1$). For this model, $p_{n,n'}$ is usually history independent –its value is fixed regardless of the operations performed beforehand–, and non symmetric –so that $p_{n,n'}$ does not have to be equal to $p_{n',n}$. The propagation model behaves as follows:*

IC.1 *Let $N_0 \subset N$ be the subset of nodes initially active, N_k be the set of active nodes at step k, and ξ be the set of ordered pairs (n, n') subjected to a propagation from n (active) toward n' (inactive). The set N_k of nodes is computed from N_{k-1}, by adding nodes as follows.*

IC.2 *We consider an active node $n \in N_{k-1}$ and an inactive node n' ($\notin N_{k-1}$) adjacent to n but whom n has still not tried to influence yet: $n' \in Ngb(n) \setminus N_{k-1}$, and $(n, n') \notin \xi$. A given node n is only offered one chance to influence each of its neighbours, and it succeeds with a probability $p_{n,n'}$; thus we add the pair (n, n') to ξ to avoid repeating the same propagation.*

IC.3 *If the adjacent node n' is successfully activated, it is added to the set of active nodes N_k.*

IC.4 *This process continues until no more activations can be performed, that is when ξ contains all the possible pairs (n, n') where n belongs to the current set of active nodes and n' is an inactive neighbour.*

The order used to choose the nodes n and their neighbours during the propagation is arbitrary.

4.1.1. Attributes

To take into account the specificities of **IC**, we need additional attributes.

First, two attributes are needed for each edge going from n to n': *Influence*, ranging on $[0, 1]$, which gives the influence probability from n on n' (i.e., $p_{n,n'}$), and *Marked*, taking for value 0 or 1, which is used to indicate whether the given pair (n, n') has already been considered, thus avoiding multiple influence tentatives. $Marked = 1$ if $(n, n') \in \xi$, 0 otherwise.

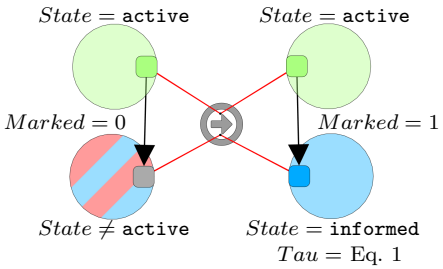
The attribute *Tau*, ranging on $[-1, 1]$, is used to measure how globally influenced a given node is. Initially, the few preset `active` nodes have their attribute $Tau = 1$, while `unaware` ones see their attribute *Tau* set to -1. During the propagation, the value of the attribute *Tau* is updated by the *influence trial* rewrite rule in order to reflect the influence probability $p_{n,n'}$, stored in the *Influence* attribute:

$$Tau = Influence - random(0, 1) \tag{1}$$

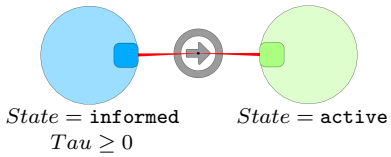
where $random(0, 1)$ is a random number in $[0, 1[$. We design the Equation 1 so that when a node is successfully influenced and ready to become active, the value of its attribute *Tau* is such that $Tau \geq 0$. This is because n' has a probability $p_{n,n'}$ of becoming active (where $p_{n,n'}$ is given as the value of the attribute *Influence*). A random number $random(0, 1)$ is thus chosen in an equi-probabilistic way and compared to the value of *Influence*. As a result, *Influence* is greater than or equal to $random(0, 1)$ in $p_{n,n'}\%$ of cases, so $Tau = Influence - random(0, 1)$ is greater or equal to 0 in $p_{n,n'}\%$ of cases.

4.1.2. Rewrite rules

The rewrite rules used to represent the **IC** model are given in Figure 8. The first one, Rule *IC influence trial* (Fig. 8(a)), shows a pair of connected nodes in the left-hand side and their corresponding replacements in the right-hand side. The node n', initially `unaware` (in `red`), or already `informed` (in `blue`) by another neighbour, is influenced –successfully or not– in the left-hand side by an `active` node n (in `green`) connected through an *unmarked* edge (its attribute *Marked* is equal to 0). In the right-hand side, n remains unchanged while n' becomes –or stays– `blue` to visually indicate that it has been *influenced* by n and `informed` of the propagation. The updated influence level Tau of n' in the right-hand side is set according to Equation 1. Furthermore, the directed edge linking the two port nodes is *marked*, by setting to 1 the attribute *Marked*. This operation is performed to limit to one the number of influence attempt for each pair of active-inactive neighbours. Inactive nodes can thus still be influenced several times but only when visited by *different* active neighbours.



(a) *IC influence trial*: influence from an `active` neighbour on an *inactive* node (either `unaware` or just `informed`).



(b) *IC activate*: an `informed` node becomes `active` when sufficiently influenced.

Figure 8: Rules used to express the Independent Cascade model (**IC**): `active` nodes are depicted in `green`, `informed` nodes in `blue` and `unaware` nodes in `red`. A bi-colour `red/blue` node can be matched to either of the two corresponding states (`unaware` or `informed`).

Rule *IC activate* in Figure 8(b) is only applied on a single node n and its objective is much more straightforward. If a node n has been sufficiently influenced, i.e., if its attribute *Tau* is greater than 0, then its state is changed, going from `informed` (`blue`) to `active` (`green`).

4.1.3. Strategy

Application of the rules describing **IC** is controlled by Strategy 4.

Strategy 4: Strategy progressive IC propagation

```
1 repeat(
2   setPos(all(property(crtGraph, node, State == active)));
3   one(IC influence trial);
4   try(one(IC activate))
5 )
```

The first instruction in the body of the repeat command (line 2) exclusively selects all the nodes whose *State* is `active` and adds them to the position P.

The second instruction then performs located rewriting (see Definition 6).[6] An `active` node will thus be used as a mandatory element from P when calling the *IC influence trial* rule (line 3) to rewrite a pair of active/inactive neighbours. At the end of the *IC influence trial* rule (Fig. 8(a)), the newly `informed` node is added to the position P.

The *IC activate* rule (Fig. 8(b)) is then immediately executed (line 4) to activate the potentially successfully influenced node. As the position P only contains the one `informed` node we have considered in the previous rule, we can ensure that the `informed` node becomes `active` as soon as it qualifies (when $Tau \geq 0$). The `try` instruction (line 4) is added to prevent any early failure if the `informed` node is not sufficiently influenced (and thus no matching `informed` node with $Tau \geq 0$ can be found). The whole process is then repeated until no more propagation can be performed (all edges are marked and all possible activations have been performed). For each rule application, the elements corresponding to the left-hand side are chosen arbitrarily among the matching possibilities.

Proposition 11. *The strategic rewrite program given by the rules in Figure 8 and Strategy 4 terminates and correctly implements the IC model.*

Proof 12. *If the initial set of active nodes is empty, the strategic program immediately terminates without changing the graph. Otherwise, at each completed iteration of the repeat loop, the set ξ of marked pairs of nodes (active, inactive) strictly increases, thanks to* IC influence trial*, and the set of active nodes N_k increases or remain constant, thanks to* IC activate*.*

Each iteration of the repeat implements a step of propagation in the model as specified in points **IC.2** *and* **IC.3** *(Definition 10): when* IC influence trial *applies, one pair of nodes (active, inactive) is chosen, the inactive node n' is influenced and the pair (n, n') is marked (i.e., added to ξ). Then the rule*

[6] Recall that a rule can only be applied if the matching subgraph contains at least one node belonging to the position P, and no element belonging to the banned set Q.

IC activate *is tried and when it applies, that is, if n' is sufficiently informed (Tau ≥ 0, which happens with probability $p_{n,n'}$, as explained in Section 4.1.1), n' becomes active and $N_k = N_{k-1} \cup \{n'\}$. Otherwise $N_k = N_{k-1}$.*

Recall that the semantics of repeat guarantees that if a command inside the body fails, the loop is terminated. The command `one`*($IC\ influence\ trial$) fails when no more unmarked pair of nodes (active, inactive) exists in the current graph. Then the repeat loop stops and the propagation is complete.*

*4.2. The linear threshold model (**LT**)*

In the **LT** model, the node activation process takes into account the neighbours' combined influence and threshold values to determine whether an informed node can become active or not (see [14] for some examples of publications describing models using activation thresholds). The model detailed below describes the first static model defined in [15].

The **LT** model seems more complicated than the **IC** model, but the approach is very similar. Here again, two different operations are used to perform the propagation: for each inactive node n', we compute the joint influence of its active neighbours, then, if the influence n' is subjected to exceed a threshold value, the node becomes active.

Definition 13 (Propagation process in LT). *Let $p_{n,n'}$ be the influence probability of n on n' ($0 \leq p_{n,n'} \leq 1$) and $\theta_{n'}$ the* threshold *value of n' –i.e., the resistance of n' to its neighbours' influence– chosen independently from n' and randomly in $[0, 1[$. Let also $S_{n'}(k)$ denote the set of nodes currently active at step k and adjacent to n', and $p_{n'}\big(S_{n'}(k)\big)$ the joint influence on n' of its active neighbours at step k. The **LT** propagation model operates as follows:*

LT.1 *Let $N_0 \subset N$ be the subset of nodes initially active and let N_k be the set of active nodes at step k. The set N_k of nodes is initialised from N_{k-1}, and computed by adding nodes as follows.*

LT.2 *An inactive node $n \notin N_{k-1}$ has its* active *neighbours' joint influence value*

computed using the formula: $p_{n'}\big(S_{n'}(k)\big) = 1 - \prod_{n \in S_{n'}(k)}(1 - p_{n,n'})$ *where* $S_{n'}(k) = Ngb(n') \cap N_{k-1}$.

LT.3 *An inactive node* n' *becomes* active *at step* k *when its neighbours' joint influence exceeds the threshold value, i.e.,* $p_{n'}\big(S_{n'}(k)\big) \geq \theta_{n'}$*, leading* n' *to be added to* N_k.

LT.4 *This process continues until, for all the joint influences up-to-date, no more activation can be performed.*

To simplify the following expressions, we note $S_{n'}$ the set $S_{n'}(k)$ being considered at the current step.

Similarly to the **IC** model, one can see that the **LT** propagation takes place in two phases: influence computation followed by activation. Before presenting the corresponding rules, we need to specify more precisely the properties of the intended propagation model from [15], as the authors present several propagation models with multiple definitions of the influence and joint influence probability of n over n' (i.e., $p_{n,n'}$ and $p_{n'}(S_{n'})$). In this paper, we are implementing the static propagation model where $p_{n,n'}$ is expressed as a constant value. Because the activation of a specific node n' is dependent of the influence probabilities coming from each of its active neighbours, we need to update their joint influence $p_{n'}(S_{n'})$ whenever one of the previously inactive neighbours of n' activates. This operation is performed using the formula $p_{n'}\big(S_{n'} \cup \{n\}\big)$, introduced in the original paper.

$$p_{n'}\big(S_{n'} \cup \{n\}\big) = p_{n'}(S_{n'}) + \big(1 - p_{n'}(S_{n'})\big) \times p_{n,n'} \tag{2}$$

This equation will add the influence of n among the other active nodes adjacent to n' (where $n \notin S_{n'}$).

4.2.1. Attributes

In order to take into account these specificities, two new attributes are needed in addition to the ones introduced earlier in **IC** (*State* and *Colour* – to define the nodes states–, *Influence* –to store the probability $p_{n,n'}$–, *Marked*

–to mark edges connecting previously visited pairs of nodes–, and *Tau* –to store the influence trial outcome). Each node is now also provided with a threshold value, stored in the attribute *Theta*, whose value is in $[0, 1]$. The joint influence probability, measuring the influence level an inactive node is subjected to, is stored using the attribute *JointInf*. Initially, the `active` nodes have their attributes $JointInf = 1$, while `unaware` ones have $JointInf = 0$. During the influence step, the value of *JointInf* on the `informed` node is updated as specified by Equation 2, which is translated to the following formula when using the appropriate attributes:

$$JointInf = JointInf_{\text{OLD}} + (1 - JointInf_{\text{OLD}}) \times Influence \tag{3}$$

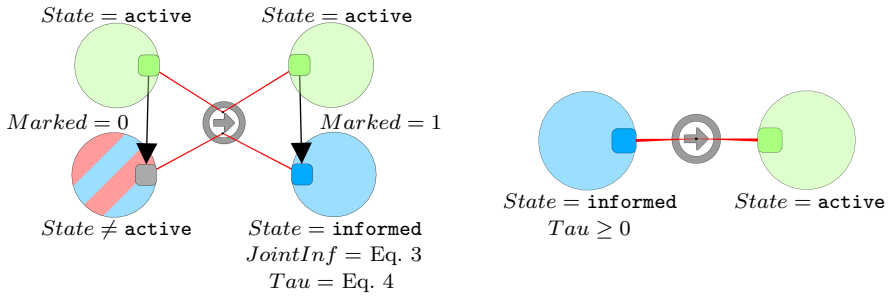
We can then compare this updated joint influence value for a node n' with its threshold value, stored in *Theta* and assign the result to the attribute Tau:

$$Tau = JointInf - Theta \tag{4}$$

If, for an `informed` node n', $Tau \geq 0$, then the joint influence of its neighbours ($JointInf$) exceeds its threshold value ($Theta$), thus leading n' to endorse the propagation subject and to activate to spread this very information to all of its neighbours.

4.2.2. Rewrite rules

The rules are quite similar to those introduced in the **IC** model. The first rule *LT influence trial* (Figure 9(a)) is applied on a connected pair of active-inactive nodes (respectively `green` and `red`/`blue`). During its application, the rule transforms an inactive node n' into an `informed` node as its `active` neighbour n tries to influence it. Two computations are performed in order to update the attributes of the inactive node n'. The attribute *JointInf* is changed during the rule *LT influence trial* by using the *Influence* value to update the previous joint influence probability measure; this is given by Equation 3. The attribute *Tau* then compares this value to the attribute *Theta* as described in Equation 4. The edge between the two nodes is marked to avoid successive influence trials



(a) *LT influence trial*: Joint influence computation from an `active` neighbour on an *inactive* node (either `unaware` or just `informed`).

(b) *LT activate*: an `informed` node becomes `active` when sufficiently influenced.

Figure 9: Rules used to express the Linear Threshold model **LT**. Colours have the same meaning as previously: `active` nodes are `green`, `informed` nodes are `blue` and `unaware` nodes are `red`. A bi-colour `red/blue` node can be in either of the two states `unaware` or `informed`.

from n to n'. As a result, n' may be influenced by several `active` nodes before their joint influence is important enough to outweigh the threshold value of n', but each of its neighbours will only be able to influence n' a single time.

The second rule, named *LT activate* (Figure 9(b)), is identical to the *IC activate* rule shown in Figure 8(b). A successfully influenced node, identified by the positive value of its *Tau* attribute, simply has its *State* attribute value set to `active`.

4.2.3. Strategy

We use the rewriting Strategy 5 to manage the rules application similarly to the **IC** model. The two strategies (used for **IC** and **LT**) differ only in the rules applied.

As in the previous model, we use a repeat command (line 1) to compute the propagation as many times as possible and start by defining a position P which will gather all the `active` nodes (line 2). One of these nodes is considered and we apply the *LT influence trial* rule (line 3) on it and one of its inactive neighbours. For the active-inactive pair of nodes (resp. n and n'), we first update the *JointInf* attribute of n' using Equation 3 and then compute its *Tau*

Strategy 5: Strategy LT propagation

```
1 repeat(
2   setPos(all(property(crtGraph, node, State == active)));
3   one(LT influence trial);
4   try(one(LT activate))
5 )
```

attribute with Equation 4. The edge connecting the two nodes is finally marked to avoid n' being repeatedly influenced by the same active node n, and n' is added to the position set P.

We then try to apply the *LT activate* rule (line 4) on an informed node whose *Tau* attribute value exceeds or equals 0. Given that the node matching the left-hand side must belong to position P, it can only be the sole `informed` node added to P, that is the inactive node which became `informed` during the *LT influence trial* rule application. If the node has not been successfully influenced (its *Tau* attribute value is lower than 0), the previously `informed` node is not an appropriate match, and thus the *LT activate* rule is not applied.

Proposition 14. *The strategic rewrite program given by the rules in Figure 9 and Strategy 5 terminates and correctly implements the LT model.*

Proof 15. *The proof is quite similar to the one proposed for **IC** (Proof 12).*

Each iteration of the repeat loop implements a step of propagation in the model as specified in points **LT.2** *and* **LT.3** *(see Definition 13): when* LT influence trial *applies, one pair (n, n') of nodes (active, inactive) is chosen and the joint influence on n' of all its active neighbours is updated (as explained in Section 4.2.1). If n' is sufficiently informed (JointInf >= Theta, i.e., Tau ≥ 0), n' becomes active and $N_{k+1} = N_k \cup \{n\}$. Otherwise $N_{k+1} = N_k$.*

*As in the case of **IC**, termination is a consequence of the fact that one of the commands in the repeat loop eventually fails: the rule* LT influence trial *can only be applied a finite number of times due to the marking of edges, hence*

`one`($LT\ influence\ trial$) *ultimately fails, the repeat loop comes to an end, and the propagation is complete.*

*4.3. Riposte (**RP**): a privacy preserving propagation model*

We propose in this section the translation of a more recent propagation model, called *Riposte*, based on the ideas introduced in [16]. Although still following the characteristic principle of randomly driven activations encountered in **IC**, **RP** differs from the two previous models on several key points. First of all, its activation and propagation mechanisms are not directly linked: both active or inactive users can propagate information, and active users are not automatically assumed to spread information to their neighbours. These features confer to **RP** the property of *plausible deniability*, which is essential to preserve the users' privacy. Indeed, independently of the user's opinions and consent concerning the propagated information, **RP** will sometimes propagate information to the user's neighbours. The user's opinion influences the probability of sharing in order to favour interesting topics, but with this propagation model, witnesses observing the exchanges within the network can now no longer precisely pinpoint which users have supported the propagation and shared the information with their neighbours. In this sense, this model improves over the previous two models. Finally, **RP** does not take into account the *influence* from one user upon another, but considers the personal *interest* a given user has in the information.

New parameters are needed to reflect these characteristics. First, let p_n be the probability given for a specific information to be re-posted by the user n. The value of p_n can be seen as a measure of how interesting the information is to n. Then, in order to prevent revealing the opinion of individual users, some randomness concerning the information propagation is incorporated. Let δ and λ be global parameters of the propagation model where $0 < \delta < 1 < \lambda$. We define $\overline{S_n}$ as the set of nodes currently unaware of the information and adjacent to n. After being informed by one of its neighbours, two different behaviours are possible. If n wishes to propagate the information (that is, n becomes active),

then either all its neighbours or none will be informed of it with a probability $\lambda/\overline{S_n}$. Alternatively, if n does not wish to propagate the information (thus, n remains "simply" informed), then the information can still be passed to all its neighbours, but this time with a weaker probability $\delta/\overline{S_n}$.

Definition 16 (Propagation process in RP). *Let p_n, λ, δ and $\overline{S_n}$ be defined as above. The complete propagation process can be described as follows:*

RP.1 *Let $N_0 \subset N$ be the subset of nodes initially aware of the information and let N_k be the set of informed nodes at step k who did not try to pass on the information to their neighbours yet. The set N_k of nodes is computed from N_{k-1} by adding and deleting nodes as follows.*

RP.2 *For a node $n \in N_{k-1}$, n tries to activate with a probability p_n.*

RP.3 *If n activates successfully, then all its neighbours are added to N_k with a probability $\lambda/\overline{S_n}$. Otherwise, for an inactive n, all its neighbours are added to N_k with a probability $\delta/\overline{S_n}$.*

RP.4 *The node n is excluded from N_k.*

RP.5 *This process continues until $N_k = \emptyset$, that is, until all the nodes that are aware of the information have been given a chance to pass the information on to their neighbours.*

As one can see, the propagation decision depends on both the user's opinion concerning the information (p_n) and the number of neighbours unaware of it ($\overline{S_n}$). In the original definition [16], the value $\overline{S_n}$ is an upper bound on the number of n's outgoing neighbours that have no knowledge yet of the information. However, a variant of the algorithm for systems where users are unable to know whether their neighbours have already heard of the information or not was also proposed. For such instance of application, which is our case, the probability is computed using the *total* number of n's outgoing neighbours instead of considering the upper bound of unaware neighbours.

4.3.1. Attributes

We naturally make use of the generic *State* and *Colour* node attributes as described in the previous models, as well as *Marked* on directed edges. But here we also need to flag nodes that have already attempted to spread the information (regardless of their activation status). This information is reflected by a new node attribute called *MarkedN*.

In addition to these, we introduce new attributes to model the specificities of **RP**. First, the attribute *Interest* records the node's interest for an information, namely the probability p_n for an information to be re-posted by n. Then the attribute *Tau* is used to store the result of the activation decision, computed as

$$Tau = Interest - random(0, 1) \tag{5}$$

where $random(0, 1)$ is a number uniformly and randomly chosen in $[0, 1[$. An `informed` node becomes `active` when $Tau \geq 0$. By default, *Tau* is set to -1 on all the nodes before the propagation begins and, as in the previous models, *Tau* is still the key attribute to enable node activation. This time however, *Tau* is computed using the *Interest* attribute instead of the *Influence* attribute as in **IC** and **LT**.

To perform the propagation according to the given parameters λ and δ of the **RP** model, an additional attribute *Share* is used to store the likeliness of n to share (i.e., spread) the information. Its value is computed as follows:

$$Share = \frac{isActive(\lambda - \delta) + \delta}{OutArity} \tag{6}$$

where *isActive* is an integer set to 1 when the attribute $State =$ `active` (and set to 0 otherwise), and *OutArity* is the cardinality of the set of outgoing neighbours. Some explanations are in order:

1. let *OutArity* be the number of outgoing edges from n;
2. if n has no neighbour to transmit the information to, then *Share* does not need to be computed; we address this issue by having *OutArity* returning -1 in such case instead of 0 to avoid errors;

3. we formulate *Share* as a single expression using λ or δ depending on the value of *isActive* (otherwise, two different rules should be used with only a small variation in the computation of *Share*).

Finally, another attribute named *Sigma* is used to store the result of the sharing decision, in a way similar to *Tau*, and is computed as

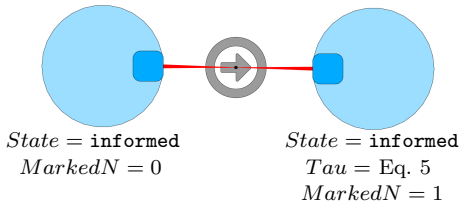
$$Sigma = Share - random(0, 1) \tag{7}$$

where $random(0, 1)$ is a random number chosen in $[0, 1[$. The propagation from n to all its neighbours is performed when the attribute *Sigma* of n is greater than or equal to 0. This attribute allows us to separate the activation process from the sharing process.

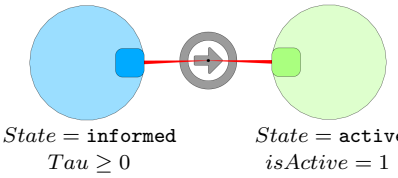
Although all these attributes are needed to emulate the propagation process, it is important to note that, in real-world applications of the **RP** algorithm, the only visible information to an external observer is whether a node has heard of the information or not. This translates to the *State* attribute marking a node as `unaware` or aware, without any distinction (such as *Colour*) between `informed` and `active` nodes.

4.3.2. Rewrite rules

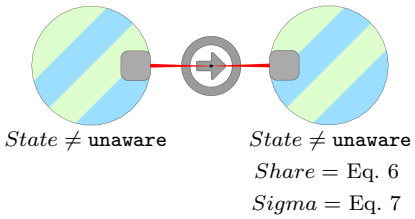
Following the formal definition of the **RP** model, we can define the following steps in the propagation mechanism given by the rules presented in Figure 10. The first rule, *RP initialisation* (Fig. 10(a)), is an opening step used to prepare the freshly `informed` nodes who did not yet try to propagate the information (i.e., unmarked nodes). A node is offered the possibility to be interested in the information, with *Tau* computed accordingly (see Equation 5), and sees its *Marked* attribute set to 1. This means that the node is soon to be considered for activation and as a candidate for spreading information. We then keep the same `informed` node and tentatively apply the rule *RP activate* (Fig. 10(b)) on it. Depending of the previously obtained *Tau* value for n, and more precisely if $Tau \geq 0$, its *activation* can take place, thus putting n in an `active` state and setting *isActive* to 1.



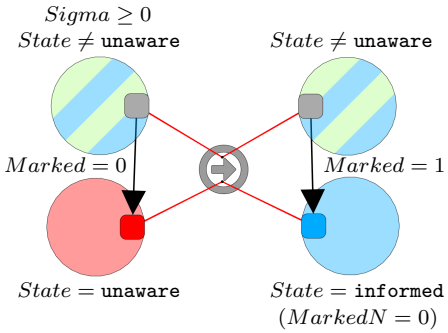
(a) *RP initialisation*: this rule is used to initialise an `informed` nodes aware of the propagation subject.



(b) *RP activate*: an `informed` node becomes `active` if its *Tau* attribute is greater or equal to 0.



(c) *RP share trial*: whether a node is `active` or `informed`, the **RP** model can decide to use it to spread the information to others.



(d) *RP inform*: a node aware of the information (`active` or `informed`), and selected to share its knowledge, informs an `unaware` neighbour.

Figure 10: Rules used to express the Riposte model **RP**. Colours keep their meaning from the previous propagation models: `active` nodes are `green`, `informed` nodes are `blue` and `unaware` nodes are `red`. A bi-colour `blue/green` node can be either `informed` or `active`.

The *RP share trial* rule, shown in Figure 10(c), computes the *Share* and *Sigma* attributes of the previously considered, and either `informed` or `active`, node n. The *Share* computation, performed following Equation 6, uses *isActive* to change the probability result depending of n's current *State*. *Sigma* then reuses the *Share* value (see Equation 7) to randomly decide whether n must share the information with its neighbours. The transmission of information to the neighbours is performed by the last rule *RP inform*, depicted in Figure 10(d). An `active` or simply `informed` node n who has been selected to propagate the

information (whose attribute $Sigma \geq 0$) informs an `unaware` neighbour n'. As a result, the `unaware` node becomes `informed`, leading it to be considered as a new potential information spreading source in the next propagation step. The newly `informed` node has its *MarkedN* attribute untouched, thus still equal to its default value (0), and ready to be subjected to the *RP initialisation* rule.

4.3.3. Strategy

The strategy used in this model is given below in Strategy 6. Much like the previous models, we use a repeat loop (line 1) in the **RP** strategy to control the rewriting steps. Recall that initially, all nodes see their attribute *MarkedN* set to 0. We initiate the strategy by choosing the node who will be the focus of rewriting in this propagation step. As we want to pick a single element and follow its behaviour, we first filter out the nodes to find a suitable candidate: among all the `informed` nodes (line 2), we select one who has never been considered to spread the propagation, that is, its *MarkedN* attribute is still equal to its default value (line 3).

Strategy 6: Riposte propagation model **RP**

```
1 repeat(
2   setPos(all(property(crtGraph, node, State == informed)));
3   setPos(one(property(crtPos, node, MarkedN == 0)));
4   one(RP initialisation);
5   try(one(RP activate));
6   one(RP share trial);
7   repeat(one(RP inform));
8 )
```

The first rule, *RP initialisation* (Fig. 10(a)), is then applied. In case not a single candidate satisfying the aforementioned conditions has be found –i.e., there is no `informed` node or all have already been considered before (with $MarkedN = 1$)– then the rule application fails and the propagation comes to an

end (line 4). However, if a matching node n exists in position P, the rule is applied on it and its *Tau* attribute is computed according to Equation 5. The rewritten node is then inserted in P and ready for the next rule application.

With our candidate node at hand, we can propose to it to endorse the propagation subject and activate. We thus try to apply the *RP activate* rule (line 5). As shown in Figure 10(b), in addition to *State* as matching attribute, *Tau* is the real filtering condition to decide whether the selected `informed` node can become `active`. This operation is optional as, in **RP**, the activation and propagation are distinct mechanisms, and thus its application, thanks to the try strategy, cannot fail. We use the *isActive* attribute to store the result of the activation trial and add the rewritten node to P.

We then apply *RP share trial* (Fig. 10(c)) on the node n (line 6) which can either be `active` or just `informed` if it did not satisfy the matching conditions of *RP activate*. The transformation computes new values for n's *Share* (Eq. 6) and *Sigma* (Eq. 7) attributes while keeping the rewritten node n in P. These values indicate to the **RP** model whether to use n as a starting point to spread the information to its neighbours.

This leads us to the nested repeat loop applying *RP inform* (Fig. 10(d)) to all neighbours (line 7). If the (indifferently `informed` or `active`) node has been selected to inform its `unaware` neighbours (n'), then its *Sigma* attribute is greater or equal to 0. The rule application sets the *State* attribute of n' to `informed` and, through its *Marked* attribute, marks the edge connecting the two nodes to avoid multiple applications of the rule on the same pair of nodes.

The newly `informed` node is now eligible to be subjected to a propagation step itself as its *MarkedN* attribute is still equal to its default value.

Proposition 17. *The strategic rewrite progam given by rules in Figure 10 and Strategy 6 terminates and correctly implements the RP model.*

Proof 18. *If the initial set of informed nodes is empty, the strategic program immediately terminates without changing the graph. Otherwise, a step of propagation is performed as specified in points* **RP.2** *and* **RP.3** *(see Definition 16):*

a single ***informed*** *node* $n \in N_{k-1}$*, which has not yet passed the information (*MarkedN $= 0$*), is selected.*

The rule RP initialisation *marks the node* n *(which ensures this node will not be considered again, as specified in point* **RP.4***, Definition 16) and updates* Tau *as explained in Section 4.3.1. Then* `try(one(`*RP activate*`))` *tries to activate* n *with a probability* p_n*, recorded in the attribute* Interest*, and reflected by the value of* Tau *as explained in Section 4.3.1.*

Whether or not the node becomes active, its Share *attribute, updated by* `one(`*RP share trial*`)`*, expresses the user's (private) opinion to be active or not, and takes into account the parameters* λ *and* δ *of the propagation model. The value of the* Sigma *attribute indicates the probability of sharing the information with all its neighbours, as specified in Definition 16, point* **RP.3***.*

When possible (if Sigma ≥ 0*), the rule* RP inform *is used to propagate the information to all unaware neighbours of the node. This is ensured by the nested* `repeat(one(`*RP inform*`))` *loop in Strategy 6 which is guaranteed to terminates thanks to the marking of the edges. Each newly informed node has their attribute* MarkedN *value equal to* 0*, and is added to* N_k *as requested in point* **RP.1***.*

The main repeat loop stops when N_k *is empty, respecting point* **RP.5***. This termination happens when all informed nodes are marked (*MarkedN $= 1$*), since this ensures failure of rule* RP initialisation *(line 4, Strategy 6).*

4.4. Adapting the Riposte model with linear thresholds

The graph rewriting formalisation highlights the key components of the different propagation models. Distinguishing the rules, in charge of local transformations, from the strategy, managing and steering their application, makes easier the comparison between the components of the different propagation models. For instance, despite their quite distinctive definitions, the basic **IC** and **LT** models were already presented as being somewhat similar, as discussed in [14]; a fact that our formalisation shows clearly: indeed, except for the *Tau* attribute computation during the *influence trial* rule application, the rules and strategies used to express the two models are identical. Expressing the models

using a common language, i.e., here the rule and strategy formalism, can thus enable experts to draw parallels between two apparently distinct models.

Additionally, breaking-down the models into smaller operations brings further benefits than just a more in depth understanding of the model. In order to properly illustrate the advantages of our approach, we show hereafter how to design a new propagation model by reusing some of the previously encountered mechanisms.

Compared to the first two models, **RP** proposes a privacy preservation feature, allowing users to influence the result of a propagation, by either endorsing or not supporting the information, without exposing their true opinion to others. We propose in the following to develop a propagation model merging elements from both **RP** and **LT**. This model, which we name *Riposte with Linear Thresholds* (**RP-LT**), hides the users' reaction to the propagation subject (endorsement or reject) while taking into account the influence from each user on its followers. As in **RP**, the users know from which of their neighbours the re-post comes from, but do not know for certain if the user is truly supportive of the propagation subject.

We recall here the different elements needed to establish the new model, keeping the notations consistent with **LT** and **RP**. An inactive node n' is influenced by each of its active neighbours n according to the probability $p_{n,n'}$ and we note $p_{n'}\big(S_{n'}(k)\big)$ the joint influence endured by n' at step k from all its active neighbours $S_{n'}(k)$. The threshold value of n', or its resistance to activation, is defined as $\theta_{n'}$. Finally, λ and δ are global parameters ($0 < \delta < 1 < \lambda$), and $\overline{S_{n'}}$ is the set of unaware nodes adjacent to n'.

Considering the core mechanism of the **LT** model, that is the necessity for a given node to be influenced multiple times to have a chance to activate, we cannot force the user to make a decision about the propagation subject after having only heard about it once. We thus add to our model an information counter. Let us define γ as the maximum number of times a node can be told an information before being asked to formulate his opinion. A node can be influenced at most γ times, but may decide to activate before.

Definition 19 (Propagation process in RP-LT). *Let $p_{n,n'}$, $p_n\big(S_n(k)\big)$, θ_n, λ, δ, γ and $\overline{S_{n'}}$ be defined as above. The model **RP-LT** is specified as follows:*

RP-LT.1 *Let $N_0 \subset N$ be the subset of nodes initially aware of the information and let N_k be the set of informed nodes at step k who did not try to pass on the information to their neighbours yet. The set N_k of nodes is computed from N_{k-1}, by adding and deleting nodes as follows.*

RP-LT.2 *For a node $n \in N_{k-1}$, n activates if the joint influence it is subjected to is at least equal to its resistance, i.e., $p_n\big(S_n(k)\big) \geq \theta_n$.*

RP-LT.3 *If n activates successfully, regardless of the number of times it has been influenced, then all its neighbours are added to N_k with a probability $\lambda/\overline{S_{n'}}$. Otherwise, for an inactive n which has been influenced γ times, all its neighbours are added to N_k with a probability $\delta/\overline{S_{n'}}$.*

RP-LT.4 *When n's neighbours, noted n', are added to N_k, their respective joint influence is updated to take into account the fact that n shares the information: $p_{n'}\big(S_{n'}(k)\big) = 1 - \prod_{n \in S_{n'}(k)}(1 - p_{n,n'})$.*

RP-LT.5 *The node n is then excluded from N_k. This process continues until $N_k = \emptyset$.*

While the core of the model description looks like **RP**'s (see Definition 16), the use of the influence ($p_{n,n'}$), joint influence ($p_n(S_n)$) and theta (θ_n) attributes distinguish the two models: where **RP** focuses on the users' interest to make the propagation evolve, **RP-LT** looks at the influence users have on one another and their response to it.

4.4.1. Attributes

For the sake of completeness, we recall the different attributes already used in **RP** and **LT**. Obviously, we keep the general attributes: *State* and *Colour* to distinguish the nodes' states, *Marked* to mark the visited pairs of nodes, as well as *MarkedN* for the nodes previously considered for activation and sharing, and

Tau to store the activation decision. We complete them with the **LT** attributes *Influence* to store $p_{n,n'}$, *Theta* for the threshold θ_n, and *JointInf* for $p_n(S_n)$.

Some **RP** attributes are also needed to complete the model: *Share* to store the node's sharing probability according to its *State*, *isActive* to mark whether the node is active or not (used to compute *Share*), *OutArity* to request the number of outgoing neighbours, and *Sigma* to store the trigger value of the propagation operation.

The equations used to compute the values of attributes *JointInf, Tau, Share, Sigma* are the same: see equations 3, 4, 6, 7 respectively.

In addition, we introduce a new attribute *Count* to track the number of times a node has been informed of the propagation subject. All nodes have the *Count* attribute initialised to 0 and each node is only given the same information γ times (from different neighbours).

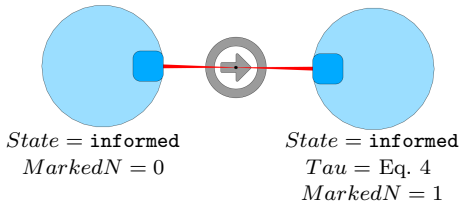
4.4.2. Rewrite rules

The rewrite rules, given in Figure 11, are quite similar to the **RP** rules. The first rule *RP-LT initialisation* (Fig. 11(a)) updates the *Tau* attribute (according to Eq. 4) of an `informed` node. When rewritten, the node stays `informed` and its *MarkedN* attribute is set to 1.

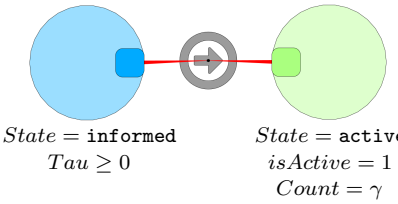
The second rule *RP-LT activate* (Fig. 11(b)) is in charge of the potential activations. When the *Tau* attribute indicates that the node n has been successfully influenced (when $Tau \geq 0$), then its *State* becomes `active` and the *isActive* attribute is accordingly updated to match n's current state. In case of activation, we also set the *Count* attribute to γ to indicate the node will no longer be responsive to influence.

Every node aware of the propagation, who either decided to activate, or who has been influenced γ times, is entitled to compute its *Share* (6) and *Sigma* (7) attribute values. This is achieved by using the *RP-LT share trial* rule (Fig. 11(c)), which applies only to nodes whose *Count* attribute exceeds or equals γ.

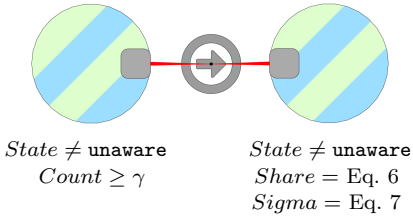
The last rule is *RP-LT inform* (Fig. 11(d)). The `active` or `informed` node



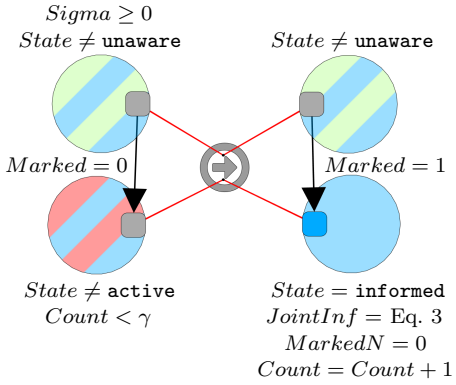
(a) *RP-LT initialisation*: the rule initialises an `informed` node, computes its *Tau* attribute and marks it using *MarkedN*.



(b) *RP-LT activate*: an `informed` node becomes `active` when its *Tau* attribute is positive.



(c) *RP-LT share trial*: whether the node is `informed` or `active`, if it has been informed γ times or made its decision, its sharing probability must be computed.



(d) *RP-LT inform*: an inactive user is informed by an `informed` or `active` neighbour. The rule computes the neighbour's *JointInf* value, resets its marker and increments its influence counter.

Figure 11: Rules used to express the Riposte with Linear Threshold model **RP-LT**.

n successfully selected to spread the information ($Sigma \geq 0$), shares it with its `unaware` or `informed` neighbours n'. To avoid multiple matching with the same pair of connected nodes, the edge between n and n' is marked. The joint influence probability of n' is updated using Equation 3 and the node is unmarked to indicate a change has happened ($MarkedN = 0$). The propagation step is only targeting inactive nodes which have been influenced less than γ times since then n' should be able to form an opinion concerning the shared

information. When n' is rewritten, its influence counter is incremented to keep track of the operation ($Count = Count + 1$).

4.4.3. Strategy

The rewriting operations are applied according to Strategy 7. Just like the strategies used for **IC** and **LT**, the ones defining the **RP** and **RP-LT** are very similar. For each rule application considered hereafter, we reinsert the newly rewritten elements in position P.

Strategy 7: Riposte with Linear thresholds propagation model **RP-LT**

```
1 repeat(
2   setPos(all(property(crtGraph, node, State == informed)));
3   setPos(one(property(crtPos, node, MarkedN == 0)));
4   one(RP-LT initialisation);
5   try(one(RP-LT activate));
6   try(one(RP-LT share trial));
7   repeat(one(RP-LT inform));
8 )
```

As for the previous models, we use a repeat loop (line 1) to perform as many propagation steps as possible. We select an `informed` node (line 2) which has not yet been subject to an initialisation or which has since undergone changes (line 3). By applying *RP-LT initialisation* (Fig. 11(a)) we mark the selected `informed` node ($MarkedN = 1$), compare the *JointInf* and *Theta* attributes and store the result in *Tau* (Eq. 4). This value is used when the strategy tries to apply the second rule *RP-LT activate* (Fig. 11(b)) to activate the node (line 4). Only successfully applied if *Tau* is positive or null, the rule transforms the `informed` node into an `active` one, respectively modifying *isActive* and *Count* values to reflect the node current *State* and indicate that its decision concerning the propagation subject has been confirmed.

The application of *RP-LT share trial* rule (Fig. 11(c)) is the only difference

between this strategy and the one proposed for **RP** (Strategy 6). Here, the *Count* attribute restricts the rule application to nodes which have either been influenced γ times or for which the joint influence is sufficient to persuade them to endorse the propagation subject (thus $Count = \gamma$ from the *RP-LT activate* rule). As a result, only when the *Count* attribute of the node in P is set to the appropriate value, will the rule successfully apply. In such a case, the *Share* and *Sigma* attributes are computed using respectively Equations 6 and 7.

Depending on its *Sigma*'s value, the node n in P will share the information with its inactive neighbours n' which have been influenced less than γ times and have not yet been contacted by n. The *RP-LT inform* rule (Fig. 11(d)) then marks the connection between n and n' and updates some of the `informed` node n' attributes: *JointInf* is recomputed taking into account the new *Influence* of n on n' (Eq. 3), the influence counter *Count* is incremented to track the new influence, and the marker *MarkedN* is reset to its default value, indicating that some changes have been applied to the attributes of n'.

To guarantee that the node in P will match with the aware node (`active` or `informed`) and not with the inactive node (`unaware` or `informed`), we use located rewriting (Definition 6). We define the aware node (`green`/`blue`) of the left-hand side as being part of the W set, thus forcing the node in position P to match with this element, ensuring the appropriate elements are rewritten as intended.

Proposition 20. *The strategic rewrite program given by rules in Figure 11 and Strategy 7 terminates and correctly implements the RP-LT model.*

Proof 21. *If the initial set of informed nodes is empty, the strategic program immediately terminates without changing the graph. Otherwise, a step of propagation is performed as specified in points* **RP-LT.2** *to* **RP-LT.4** *(Definition 19): a single* ***informed*** *node $n \in N_{k-1}$, which has not yet passed the information (*$\text{MarkedN} = 0$*), is selected.*

The rule RP-LT initialisation *marks the node and updates* Tau *as explained in Section 4.4.1 (step* **RP-LT.1***). Then* `try(one(`*RP-LT activate*`))` *tries to*

activate it, according to the value of Tau *as explained in Section 4.3.1 (step* **RP-LT.2***).*

Whether or not the node becomes active, its Share *and* Sigma *attributes are updated by* `one(`*RP-LT share trial*`)` *(step* **RP-LT.2***), provided it has been influenced γ times or is active but did not yet propagate the information (step* **RP-LT.3***).*

The rule RP-LT inform *propagates the information to an unaware neighbour of the node. Thanks to the marking of edges, the command* `repeat(one(`*RP inform*`))` *terminates, and each newly informed node is unmarked (*MarkedN $=$ 0*) and added to N_k (step* **RP-LT.1***).*

The main repeat loop stops when N_k is empty (step **RP-LT.5***), i.e., when all the informed nodes are marked (*MarkedN $= 1$*).*

Termination of the ***RP-LT*** *strategy is a consequence of the fact that the strategy eventually run out of suitable nodes to apply* RP-LT initialisation *on, that is $N_k = \emptyset$. The rule updates the* MarkedN *attribute value which will only be reset to* 0 *in* RP-LT inform *when a new influence touch the node; this operation however can only happen a limited number of times (set by γ's value). Afterward, the* MarkedN *attribute will no longer be modified thus leaving the node as unmatchable for future applications. We ensure that the command* `one(`*RP-LT initialisation*`)` *(line 4) in the body of the repeat will fail, hence that the program will terminate. We also use the marking of edges in rule* RP-LT inform *to guarantee that each connected pair is only considered once. This ensures that the nested repeat loop comes to an end when all inactive neighbours have been visited (thus leaving* RP-LT inform *to fail).*

5. A framework to study social networks

At this point, we may argue that we have the ingredients for a social network modelling framework:

- in Section 3, we have explained how to generate social network models tailored to various sizes, number of links and communities. The capability of

generating arbitrary models is important to validate new methods or algorithms and check their behaviour.

- in Section 4, we have formalised with labelled port graphs, rules and strategies, three known models of propagation with different properties. We identified common basic features (attributes, rules, strategies) and simultaneously better understood what is different between these three approaches. For instance, we have identified that the obfuscation property is ensured by the addition of new node attributes and rules that reflect intermediate state changes. This formalisation then helped us to design a new propagation model **RP-LT**, by combining the features of the **LT** and **RP** models.

- The formalism used, based on labelled port graphs, rewrite rules and strategies, provides the logical background necessary to understand and analyse the programs and their executions. For instance we have shown how to prove the correctness of the different strategic rewrite programs for each propagation model.

- Visualisation features provided in the PORGY environment are indeed an important component of the framework. The prototyping aspect of rules and strategies is amplified by visualisation of results, especially for large graphs. For instance, the result of generating a social network according to different parameters is illustrated quite well on examples of Section 3. Then the behaviour of a new propagation model like **RP-LT** can be experimented and visualised on a generated network with a selected topology.

- The environment can for instance be used to compare two propagation models in the same line as done in [8]. Visualisation provides a first intuition for comparing two models applied to the same starting network, but indeed comparison needs to use measurement methods appropriate to propagation phenomena in social networks. This is left for future work.

6. Conclusion

Our first experiments and results on generation and propagation in social networks illustrate how labelled port graph strategic rewriting provides a suitable common formalism in which different mathematical models can be expressed and compared. In Section 3, we have shown the power of topological matching and the importance of strategies, while in Section 4, the attributes have been heavily used to steer the rewriting operations. In both sections, rules are quite simple but apply to big graphs.

For the social network community, the rewrite rule approach is not quite surprising because some works such as [36] already use rules to generate social networks, although without claiming it. Expressing different models in the same formalism facilitates the comparison of algorithms and models. With the development of social networks analysis, there are many opportunities where simulations can indeed be of assistance during decision taking, for instance to prevent bad situations, to test counter-measures, or to look for optimal diffusion strategy. Although we did not develop this aspect here, when modelling the evolution of a network, the derivation tree (also a port graph) provides support for history tracking, state comparison, state recovery and backtracking.

Overall, several issues remain to be addressed. Although graph rewriting has been largely studied, social network applications have only recently been developed, and require a drastic change of scale. Dealing with millions of nodes and edges requires great attention to size and complexity. There is also room for improvement in data storage and retrieval (relevant for graph data bases), subgraph matching algorithms –either exact or approximate– for finding one or all solutions, parallel graph rewriting avoiding dangling edges, and probabilistic or stochastic issues for matching and rewriting, for instance, in the context of imprecise data or privacy constraints.

Also related to size, but even more to complexity of data, there is a need for data structuring and management, that may be carried on by abstraction pattern, focusing on points of interests, hierarchies and views (for instance, through

multi-layer graphs). All these notions need a precise and logical definition that may be influenced by well-known programming language concepts.

Like programs, data need certification and validation tools and processes, not only at a single step but all along their evolution. The knowledge developed in the logic and rewriting community should be valuable in this context.

This study has also revealed the importance of visualisation and raises some challenges in this area. Visualisation is important, more widely, for data analysis, program engineering, program debugging, testing or verifying. However, the representation of dynamic or evolving data, such as social networks or richer graph structures, is yet an actual research topic for the visualisation community.

Acknowledgements. We thank Guy Melançon (University of Bordeaux) and all the other members of the PORGY project.

References

[1] P. Carrington, J. Scott, S. Wasserman, Models and Methods in Social Network Analysis, Structural Analysis in the Social Sciences, Cambridge University Press, 2005.
URL http://books.google.fr/books?id=4Ty5xP_KcpAC

[2] M. Newman, A.-L. Barabási, D. J. Watts, The structure and dynamics of networks, Princeton Studies in Complexity, Princeton University Press, 2006.

[3] J. Scott, P. J. Carrington, The SAGE Handbook of Social Network Analysis, SAGE, 2011.

[4] M. Granovetter, Threshold models of collective behavior, American Journal of Sociology 83 (6) (1978) 1420.
URL https://sociology.stanford.edu/publications/threshold-models-collective-behavior

[5] P. Dodds, D. Watts, A generalized model of social and biological contagion, Journal of Theoretical Biology 232 (4) (2005) 587 – 604. doi:http://dx.doi.org/10.1016/j.jtbi.2004.09.006.
URL http://www.sciencedirect.com/science/article/pii/S0022519304004515

[6] E. Bertuzzo, R. Casagrandi, M. Gatto, I. Rodriguez-Iturbe, A. Rinaldo, On spatially explicit models of cholera epidemics, Journal of The Royal Society Interface 7 (43) (2010) 321–333. arXiv:http://rsif.royalsocietypublishing.org/content/7/43/321.full.pdf+html, doi:10.1098/rsif.2009.0204.
URL http://rsif.royalsocietypublishing.org/content/7/43/321.abstract

[7] W. Chen, C. Wang, Y. Wang, Scalable influence maximization for prevalent viral marketing in large-scale social networks, in: Proc. of the 16^{th} ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, KDD '10, ACM, 2010, pp. 1029–1038. doi:10.1145/1835804.1835934.
URL http://doi.acm.org/10.1145/1835804.1835934

[8] J. Vallet, H. Kirchner, B. Pinaud, G. Melançon, A visual analytics approach to compare propagation models in social networks, in: A. Rensink, E. Zambon (Eds.), Proc. Graphs as Models, GaM 2015, Vol. 181 of EPTCS, 2015, pp. 65–79. doi:10.4204/EPTCS.181.5.
URL http://dx.doi.org/10.4204/EPTCS.181.5

[9] M. Fernández, H. Kirchner, B. Pinaud, J. Vallet, Labelled Graph Rewriting Meets Social Networks, in: D. Lucanu (Ed.), Rewriting Logic and Its Applications, WRLA 2016, Vol. 9942 of LNCS, Springer International Publishing Switzerland, Eindhoven, Netherlands, 2016, pp. 1–25. doi:10.1007/978-3-319-44802-2_1.
URL https://hal.inria.fr/hal-01347355

[10] O. Andrei, M. Fernández, H. Kirchner, G. Melançon, O. Namet, B. Pinaud, PORGY: Strategy-Driven Interactive Transformation of Graphs, in: R. Echahed (Ed.), 6^{th} Int. Workshop on Computing with Terms and Graphs, Vol. 48, 2011, pp. 54–68. doi:10.4204/EPTCS.48.7.
URL http://hal.inria.fr/inria-00563249/en

[11] M. Fernández, H. Kirchner, O. Namet, A strategy language for graph rewriting, in: G. Vidal (Ed.), Logic-Based Program Synthesis and Transformation, Vol. 7225 of LNCS, Springer Berlin Heidelberg, 2012, pp. 173–188. doi:10.1007/978-3-642-32211-2_12.
URL http://dx.doi.org/10.1007/978-3-642-32211-2_12

[12] M. Fernández, H. Kirchner, B. Pinaud, Strategic port graph rewriting: An interactive modelling and analysis framework, in: D. Bosnacki, S. Edelkamp, A. Lluch-Lafuente, A. Wijs (Eds.), Proc. 3rd Workshop on GRAPH Inspection and Traversal Engineering, GRAPHITE 2014, Vol. 159 of EPTCS, 2014, pp. 15–29. doi:10.4204/EPTCS.159.3.
URL http://dx.doi.org/10.4204/EPTCS.159.3

[13] M. Fernández, H. Kirchner, B. Pinaud, Strategic Port Graph Rewriting: An Interactive Modelling and Analysis Framework, Research report, Inria (Jan. 2016).
URL https://hal.inria.fr/hal-01251871

[14] D. Kempe, J. Kleinberg, É. Tardos, Maximizing the spread of influence through a social network, in: Proc. of the 9^{th} ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, KDD '03, ACM, 2003, pp. 137–146. doi:10.1145/956750.956769.
URL http://doi.acm.org/10.1145/956750.956769

[15] A. Goyal, F. Bonchi, L. V. Lakshmanan, Learning influence probabilities in social networks, in: Web Search and Data Mining, Proc. of the 3rd ACM Int. Conf. on, WSDM '10, ACM, 2010, pp. 241–250. doi:10.1145/

1718487.1718518.
URL http://doi.acm.org/10.1145/1718487.1718518

[16] G. Giakkoupis, R. Guerraoui, A. Jégou, A.-M. Kermarrec, N. Mittal, Privacy-Conscious Information Diffusion in Social Networks, in: Y. Moses, M. Roy (Eds.), DISC 2015, Vol. LNCS 9363 of 29^{th} Int. Symp. on Distributed Computing, Toshimitsu Masuzawa and Koichi Wada, Springer-Verlag Berlin Heidelberg, Tokyo, Japan, 2015. doi:10.1007/978-3-662-48653-5_32.
URL https://hal.archives-ouvertes.fr/hal-01207162

[17] B. Pinaud, G. Melançon, J. Dubois, PORGY: A Visual Graph Rewriting Environment for Complex Systems, Computer Graphics Forum 31 (3) (2012) 1265–1274. doi:10.1111/j.1467-8659.2012.03119.x.
URL http://hal.inria.fr/hal-00682550

[18] H. Barendregt, M. van Eekelen, J. Glauert, J. R. Kennaway, M. Plasmeijer, M. Sleep, Term graph rewriting, in: Proc. of PARLE, Parallel Architectures and Languages Europe, no. 259-II in LNCS, Springer-Verlag, 1987, pp. 141–158.

[19] Y. Lafont, Interaction nets, in: Proc. of the 17^{th} ACM Symp. on Principles of Programming Languages (POPL'90), ACM Press, 1990, pp. 95–108.

[20] K. Barthelmann, How to construct a hyperedge replacement system for a context-free set of hypergraphs, Tech. rep., Universität Mainz, Institut für Informatik (1996).

[21] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, M. Löwe, Algebraic approaches to graph transformation - part i: Basic concepts and double pushout approach, in: Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations, World Scientific, 1997, pp. 163–246.

[22] D. Plump, Term graph rewriting, in: H. Ehrig, G. Engels, H.-J. Kreowski, G. Rozenberg (Eds.), Handbook of Graph Grammars and Computing by Graph Transformations, Volume 2: Applications, Languages, and Tools, World Scientific, 1998, pp. 3–61.

[23] A. Habel, J. Müller, D. Plump, Double-pushout graph transformation revisited, Mathematical Structures in Computer Science 11 (5) (2001) 637–688.

[24] O. Andrei, H. Kirchner, A Rewriting Calculus for Multigraphs with Ports., in: Proc. of RULE'07, Vol. 219 of Electronic Notes in Theoretical Computer Science, 2008, pp. 67–82.

[25] O. Andrei, H. Kirchner, A Higher-Order Graph Calculus for Autonomic Computing, in: Graph Theory, Computational Intelligence and Thought. Golumbic Festschrift, Vol. 5420 of LNCS, Springer, 2009, pp. 15–26.

[26] P. Borovanský, C. Kirchner, H. Kirchner, P.-E. Moreau, C. Ringeissen, An overview of ELAN, Electronic Notes in Theoretical Computer Science 15 (1998) 55–70.

[27] D. Plump, The Graph Programming Language GP, in: S. Bozapalidis, G. Rahonis (Eds.), CAI, Vol. 5725 of LNCS, Springer, 2009, pp. 99–122.

[28] P. Erdős, A. Rényi, On the evolution of random graphs, in: Publication of the Mathematical Institute, Vol. 5, Hungarian Academy of Sciences, 1960, pp. 17–61.

[29] D. J. Watts, S. H. Strogatz, Collective dynamics of 'small-world' networks, Nature 393 (1998) 440–442. doi:10.1038/30918.
URL http://dx.doi.org/10.1038/30918

[30] A.-L. Barabási, R. Albert, Emergence of scaling in random networks, Science 286 (5439) (1999) 509–512. arXiv:http://science.sciencemag.org/content/286/5439/509.full.pdf, doi:10.1126/science.286.5439.509.
URL http://science.sciencemag.org/content/286/5439/509

[31] L. Wang, F. Du, H. P. Dai, Y. X. Sun, Random pseudofractal scale-free networks with small-world effect, The European Physical Journal B - Condensed Matter and Complex Systems 53 (3) (2006) 361–366. doi:10.1140/epjb/e2006-00389-0.
URL http://dx.doi.org/10.1140/epjb/e2006-00389-0

[32] V. Batagelj, U. Brandes, Efficient generation of large random networks, Phys. Rev. E 71 (2005) 036113. doi:10.1103/PhysRevE.71.036113.
URL http://link.aps.org/doi/10.1103/PhysRevE.71.036113

[33] U. Brandes, D. Wagner, Analysis and visualization of social networks, in: M. Jünger, P. Mutzel (Eds.), Graph Drawing Software, Mathematics and Visualization, Springer Berlin Heidelberg, 2004, pp. 321–340. doi:10.1007/978-3-642-18638-7_15.
URL http://dx.doi.org/10.1007/978-3-642-18638-7_15

[34] M. Kivelä, A. Arenas, M. Barthelemy, J. P. Gleeson, Y. Moreno, M. A. Porter, Multilayer networks, Journal of Complex Networks 2 (3) (2014) 203–271. arXiv:http://comnet.oxfordjournals.org/content/2/3/203.full.pdf+html, doi:10.1093/comnet/cnu016.
URL http://comnet.oxfordjournals.org/content/2/3/203.abstract

[35] S. Milgram, The small world problem, Psychology Today 2 (1967) 60–67.

[36] N. Kejžar, Z. Nikoloski, V. Batagelj, Probabilistic inductive classes of graphs, The Journal of Mathematical Sociology 32 (2) (2008) 85–109. arXiv:http://dx.doi.org/10.1080/00222500801931586, doi:10.1080/00222500801931586.
URL http://dx.doi.org/10.1080/00222500801931586

[37] D. Cartwright, F. Harary, Structural balance: a generalization of Heider's theory, Psychological Review 63 (1956) 277–293. doi:10.1037/h0046049.

[38] J. Leskovec, D. Huttenlocher, J. Kleinberg, Signed networks in social media, in: Proc. of the SIGCHI Conf. on Human Factors in Computing Systems,

CHI '10, ACM, 2010, pp. 1361–1370. doi:10.1145/1753326.1753532.
URL http://doi.acm.org/10.1145/1753326.1753532

[39] B. Nick, C. Lee, P. Cunningham, U. Brandes, Simmelian backbones: Amplifying hidden homophily in facebook networks, in: Advances in Social Networks Analysis and Mining (ASONAM), 2013 IEEE/ACM Int. Conf. on, 2013, pp. 525–532. doi:10.1145/2492517.2492569.

[40] W. Chen, A. Collins, R. Cummings, T. Ke, Z. Liu, D. Rincón, X. Sun, Y. Wang, W. Wei, Y. Yuan, Influence maximization in social networks when negative opinions may emerge and propagate, in: Proc. of the 11^{th} SIAM Int. Conf. on Data Mining, SDM 2011, 2011, pp. 379–390. doi:10.1137/1.9781611972818.33.
URL http://dx.doi.org/10.1137/1.9781611972818.33

[41] L. Wonyeol, K. Jinha, Y. Hwanjo, CT-IC: Continuously activated and time-restricted independent cascade model for viral marketing, in: Data Mining (ICDM), 2012 IEEE 12^{th} Int. Conf. on, 2012, pp. 960–965. doi:10.1109/ICDM.2012.40.

[42] D. Kempe, J. Kleinberg, É. Tardos, Influential nodes in a diffusion model for social networks, in: L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, M. Yung (Eds.), Automata, Languages and Programming, Vol. 3580 of LNCS, Springer Berlin Heidelberg, 2005, pp. 1127–1138. doi:10.1007/11523468_91.
URL http://dx.doi.org/10.1007/11523468_91

[43] D. J. Watts, A simple model of global cascades on random networks, Proc. of the National Academy of Sciences 99 (9) (2002) 5766–5771. arXiv:http://www.pnas.org/content/99/9/5766.full.pdf+html, doi:10.1073/pnas.082090499.
URL http://www.pnas.org/content/99/9/5766.abstract

[44] M. Gomez-Rodriguez, J. Leskovec, A. Krause, Inferring networks of diffusion and influence, in: Proc. of the 16^{th} ACM SIGKDD Int. Conf. on

Knowledge Discovery and Data Mining, KDD '10, ACM, 2010, pp. 1019–1028. doi:10.1145/1835804.1835933.
URL http://doi.acm.org/10.1145/1835804.1835933